An Efficient ECDSA-based Adaptor Signature

Binbin Tu, Yu Chen, and Min Zhang

School of Cyber Science and Technology, Shandong University, Qingdao, China

Abstract. Adaptor signature can tie together transaction authorization and witness extraction and has become an important tool for solving the scalability and interoperability problems in the blockchain. Aumayr et al. first provide the formalization of the adaptor signature and present a provably secure ECDSA-based adaptor signature. However, their scheme requires zero-knowledge proof in the pre-signing phase to ensure the signer works correctly which leads to low efficiency.
In this paper, we construct efficient ECDSA-based adaptor signature schemes and give security proof based on ECDSA. The pre-signing algorithm of our schemes is similar to the ECDSA signing algorithm except for modifying some parameters by using the pre-signing public parameter and the instance as the verification key and base point. Therefore, our schemes can completely reuse the implementation of ECDSA and enjoy the same efficiency as ECDSA. In particular, considering special verification scenarios, the pre-signature of our scheme can even reduce the number of zero-knowledge proofs which further improves the efficiency. Last, we conduct an experimental evaluation, demonstrating that the performance of our ECDSA-based adaptor signature is similar to ECDSA and is comparable to the state-of-the-art ECDSA-based adaptor signature.

Keywords: Adaptor signature, ECDSA, ECDSA-based adaptor signature

1 Introduction

Adaptor signatures (AS), also known as scriptless scripts, are introduced by Poelstra [1] and recently formalized by Aumayr et al. [2] AS can be seen as an extension over a digital signature with respect to an instance of a hard relation. Namely, the signer can generate a pre-signature with the pre-signing key, message, and an instance of a hard relation, such that the pre-signature can be adapted into a valid full signature by using the witness of the hard relation. The full signature can be verified in the same way as the original verification algorithm. What's more, the witness can be extracted by using the pre-signature and the full signature. Therefore, AS can provide the following properties: (i) only the signer knowing the pre-signing key can generate the pre-signature; (ii)only the user knowing the witness of the hard relation can convert the pre-signature into a valid full signature; (iii) anybody can check the validity of the pre-signature and the corresponding signature and use them to extract the witness of the hard relation.

Tying together the signature and witness extraction, AS brings about various advantages of reducing the operations on-chain and supporting advanced functionality beyond the limitation of the blockchain's scripting language. AS has been shown highly useful in many blockchain applications such as payment channels [2,3,4,5,6], payment routing in payment channel networks (PCNs) [7,8,9,10], atomic swaps [11,12,10], and many others.

Poelstra [1] first gives a Schnorr-based AS that is limited to cryptocurrencies using Schnorr signatures [13] and thus is not compatible with those systems, prominently Bitcoin. Then, Moreno-Sanchez and Kate [14] present an ECDSA-based AS and its two-party version, but their schemes are not clear how to prove security. Malavolta et al. [8] present protocols for two-party adaptor signatures based on ECDSA [15]. However, they do not define AS as a stand-alone primitive and formalize the security definition for the threshold primitive and hence the security of their schemes has not been analyzed completely, such as the lack of the witness extractability. Until Aumayr et al. [2] first formalize AS as a standalone primitive and prove the security of their ECDSA-based AS based on the strong unforgeability of positive ECDSA in the Universal Composability (UC) framework [16]. They exquisitely modify the hard relation in [14], by adding zero-knowledge proof such that they can extract the witness in the random model [17], namely "self-proving structure". However, their ECDSA-based AS requires another zero-knowledge proof in the pre-signing phase to ensure that the signer works correctly, which leads to low efficiency.

1.1 Our Contributions

In this paper, we propose an ECDSA-based AS (ECDSA-AS) and use "self-proving structure" [2] $I_Y = (Y, \pi_Y)$ to prove the security based on positive ECDSA in UC framework [16]. And then, we observe that the zero-knowledge proof used in the pre-signing phase is independent of the message and random number, so we can compute the pre-signing public parameter and zero-knowledge proof offline to improve the efficiency of the online pre-signing phase.

Based on above observation, we construct two efficient ECDSA-AS schemes. For convenience, we denote our two efficient ECDSA-AS schemes by ECDSA-AS1 and ECDSA-AS2. ECDSA-AS1 uses the pre-signing key x as a witness to prove the hard relation $((G, Q = xG, Y, Z = xY), x)$ satisfies equality of discrete logarithms. ECDSA-AS2 uses the witness y of hard relation (I_Y, y) as a witness to prove the hard relation $((G, Y = yG, Q, Z = yQ), y)$ satisfies equality of discrete logarithms. The online pre-signing algorithm of our ECDSA-AS1/2 is similar to the original ECDSA signing algorithm except for modifying some parameters by using (Z, Y) as verification key and base point instead of (Q, G) used in ECDSA. Therefore, our schemes can enjoy the same efficiency and functionality extension as ECDSA.

Benefiting from the structure of our ECDSA-AS1/2, considering special verification scenarios in which only the pre-signers check the validity of pre-signatures, such as in the atomic swaps two parties pre-signing and verify the pre-signatures off-chain, the zero-knowledge proof used in pre-signing phase can be reduced, which further improves the efficiency. We briefly show the main techniques as follows:

ECDSA-based adaptor signature. ECDSA-AS can be seen as an extension of ECDSA with a hard relation $(I_Y = (Y = yG, \pi_Y), y)$, where $\pi_Y \leftarrow \mathsf{P}_Y(Y, y)$, P_Y denotes the proving algorithm in the zero-knowledge proof system [1]. We briefly introduce our ECDSA-AS as follows: $(Q = xG, x)$ denotes the verification key and signing key. The signer computes a pre-signing public parameter $Z = xY$, a zero-knowledge proof $\pi_Z \leftarrow \mathsf{P}_Z((G, Q, Y, Z), x)$ [2], and then chooses a random number $k \leftarrow \mathbb{Z}_q$, and computes $r = f(KY)$ [3], $\hat{s} = k^{-1}(h(m) + rx) \bmod q$, and outputs the pre-signature $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$. The verification algorithm checks the validity of π_Z and $r \overset{?}{=} f(\hat{s}^{-1} \cdot h(m) \cdot Y + \hat{s}^{-1} \cdot r \cdot Z))$. The adaptor algorithm takes the witness y and the pre-signature $\hat{\sigma}$ as inputs and computes $s = \hat{s} \cdot y^{-1} \bmod q = k^{-1}y^{-1}(h(m) + rx) \bmod q$, and outputs ECDSA signature $\sigma = (r, s)$. The extraction algorithm can extract the witness from ECDSA signature and pre-signature by computing $y = \hat{s}/s$.

We use "self-proving structure" $(I_Y = (Y = yG, \pi_Y), y)$ following [2] to ensure provable security. Intuitively speaking, in the security proof, because the zero-knowledge proof system holds straight-line extractor, the simulator can extract the witness y from the instance $I_Y = (Y = yG, \pi_Y)$, then it can take advantage of the ECDSA signing oracle to simulate the pre-signing oracle.

Offline/Online. In [2], the signer computes the pre-signing public parameter $K = kY$ and proves the hard relation $((G, \hat{K}, Y, K), k)$ satisfies equality of discrete logarithms $\pi_K \leftarrow \mathsf{P}((G, \hat{K}, Y, K), k)$, that is, there exists the witness k that is a random number used in the pre-signing algorithm, such that $\hat{K} = kG$ and $K = kY$. In ECDSA-AS1, the signer computes the pre-signing public parameter $Z = xY$ and proves the hard relation $((G, Q, Y, Z), x)$ satisfies equality of discrete logarithms $\pi_Z \leftarrow \mathsf{P}_Z((G, Q, Y, Z), x)$, that is, there exists the witness x that is pre-signing key, such that $Q = xG$ and $Z = xY$. According to the structure $Z = xY = yQ$, in ECDSA-AS2, the signer computes the pre-signing public parameter $Z = yQ$ and proves the hard relation $((G, Y, Q, Z), y)$ satisfies equality of discrete logarithms $\pi_Z \leftarrow \mathsf{P}_Z((G, Y, Q, Z), y)$, that is, there exists the witness y that is the witness of hard relation $(I_Y = (Y, \pi_Y), y)$, such that $Y = yG$ and $Z = yQ$.

Compared with [2], the pre-signing public parameter Z and the proof π_Z in our ECDSA-AS are independent of the message m and the random number k used in the pre-signature, so the part can be executed offline. At the same time, for same hard relation $(I_Y = (Y, \pi_Y), y)$, one zero-knowledge proof for $((G, Q, Y, Z), x)$ or $((G, Y, Q, Z), y)$ can be used to pre-signing many messages, but due to using the random number k as the witness in [2], the size of zero-knowledge proof is linearly related to the number of pre-signatures. In particular,

[1] The zero-knowledge proof system requires straight-line extractor, also namely online extractor [17].

[2] This zero-knowledge proof system does not require straight-line extractor.

[3] The function f is defined as the projection to x-coordinate.

in ECDSA-AS2, all pre-signing public parameters Z_i and proofs π_{Z_i} can be generated offline and batched for all signers U_i by the hard relation chooser who has the witness y.

Special verification scenario. According to the definition of AS [2], the pre-signature can be verified by anyone. However, in some applications, AS does not require such a strong property, such as the atomic swaps, the pre-signature is only verified by the participants of the protocol. To be specific, two users U_0 and U_1 want to exchange two different cryptocurrencies c_0 and c_1. U_0 (hard relation chooser) chooses a hard relation (I_Y, y) and sends I_Y to U_1; U_0 computes a pre-signature $\hat{\sigma}_0$ for spending c_0 to U_1; U_1 computes a pre-signature $\hat{\sigma}_1$ for spending c_1 to U_0; Both parties can check the validity of the pre-signature from each other; Then, U_0 can compute the full signature σ_1 and gets c_1 by publishing σ_1 on blockchain; U_1 can extract the witness y from $\hat{\sigma}_1$ and σ_1, and then computes the full signature σ_0 and gets c_0 by publishing σ_0 on blockchain.

As we can see, in above atomic swaps, the pre-signatures do not need to be published on the blockchain, and only verified by participants U_0 and U_1. ECDSA-AS1/2 can reduce some zero-knowledge proofs in this kind of special verification scenario. That is, the zero-knowledge proof of the pre-signing public parameter Z_1 of U_1 can be reduced, because of U_0 can use the witness y to check the structure of $Z_1 = yQ_1$ without proof π_{Z_1}, but the zero-knowledge proof π_{Z_0} of U_0 cannot be removed.

Performance. Benefiting from holding the original ECDSA signing structure, our schemes can enjoy the same efficiency and completely reuse the hardware and software implementation of ECDSA except for modifying parameters, so it is friendly to upgrade existing ECDSA application. Then, We recall ECDSA-AS [14,2], and show the theoretical analysis of our ECDSA-AS. Our schemes can achieve the same level of efficiency as ECDSA and are more efficient than the state-of-the-art ECDSA-AS [14,2], in particular, ECDSA-AS1/2 only computes once point multiplication operation, while ECDSA-AS in [14,2] need four times point multiplication operation in online pre-signing phase. Last, we realize our schemes based on the OpenSSL, and the running times of all algorithms are the level of microseconds. The experimental results show that our proposed schemes are practical.

Applications. We develop atomic swaps to batched atomic swaps in which one user U_0 can exchange different cryptocurrencies with many users U_i in a single batch. U_0 spends c_{0i} to U_i and U_i spends c_{1i} to U_0. Batched atomic swaps apply to one party with multiple addresses (accounts) or one party with a lot of transactions that needs to exchange with multiple users once, such as the exchange. For one hard relation, U_0 can compute and send a batch of pre-signatures σ_{0i} for each users U_i, after checking the validity of all pre-signatures, each user U_i can compute the pre-signatures σ_{1i} and sends it to U_0, and then U_0 can check the validity of all pre-signatures and computes signatures σ_{1i} and gets c_{1i} by publishing signatures σ_{1i} on blockchain. Last, each user can get signatures

σ_{1i} and extract the witness y, and computes the signature σ_{0i} and gets c_{0i} by publishing signatures σ_{0i} on blockchain.

Our ECDSA-AS2 is more suitable for applying to above batched atomic swaps, because that U_0 can compute the pre-signing public parameters $Z_i = yQ_i = x_iY$ for other users U_i without zero-knowledge proofs π_{Z_i} in the off-line phase. Each user U_i can check $Z_i = x_iY$ and run original ECDSA signing algorithm by using (Z_i, Y) as verification key and base point instead of (Q_i, G). Therefore, the on-line pre-signing algorithm of ECDSA-AS2 is same to ECDSA signing algorithm. What's more, for a batch of transactions, U_0 uses same hard relation Y and computes same pre-signing public parameters $Z_0 = yQ_0 = x_0Y$ and same zero-knowledge proof π_{Z_0} for all other users.

2 Preliminaries

2.1 Notations

For $n \in \mathbb{N}$, 1^λ denotes the string of λ ones. Throughout, we use λ to denote the security parameter. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. We denote a probabilistic polynomial-time algorithm by PPT. If S is a set then $s \leftarrow S$ denotes the operation of sampling an element s of S at random.

3 Signature Scheme

A signature scheme consists of three algorithms $\sum = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ defined as follows.

- $\mathsf{Gen}(1^\lambda) \to (vk, sk)$. The key generation algorithm takes the security parameter as input and outputs a verification key vk and a secret key sk.
- $\mathsf{Sign}_{sk}(m) \to \sigma$. The signing algorithm takes the secret key sk and the message $m \in \{0,1\}^*$ as input, and outputs a signature σ.
- $\mathsf{Vrfy}_{vk}(m, \sigma) \to 0/1$. The verification algorithm takes the verification key vk, the message $m \in \{0,1\}^*$, and the signature σ as input, and outputs either 0 or 1.

The correctness is that for any $(vk,\ sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and $m \in \{0,1\}^*$, we have $\mathsf{Vrfy}_{vk}(m,\ \mathsf{Sign}_{sk}(m)) \to 1$.

Definition 1 (SUF-CMA security). *A signature scheme $\sum$ is SUF–CMA secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that:* $\Pr[sSigForge_{\mathcal{A},\sum}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where the experiment sSigForge$_{\mathcal{A},\sum}$ is defined as follows:*

$\mathrm{sSigForge}_{\mathcal{A},\sum}(\lambda)$	$\mathcal{O}_{\mathrm{Sign}_{sk}}(m)$
$(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$\sigma \leftarrow \mathsf{Sign}_{sk}(m)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Sign}_{sk}}(\cdot)}(vk)$	$\mathcal{Q} = \mathcal{Q} \cup \{m, \sigma\}$
$\mathrm{return}((m^*, \sigma^*) \notin \mathcal{Q} \wedge \mathsf{Vrfy}_{vk}(m^*, \sigma^*))$	$\mathrm{return}\ \sigma$

3.1 Adaptor Signature Scheme

An adaptor signature scheme [2] w.r.t. a hard relation $\mathsf{R} = \{Y, y\}$ and a signature scheme $\sum = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of four algorithms $\Pi_{\mathsf{R},\sum} =$ $(\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Ext})$ defined as:

- $\mathsf{pSign}_{sk}(m, Y) \rightarrow \hat{\sigma}$: on input a pre-signing key sk, an instance Y and a message $m \in \{0,1\}^*$, outputs a pre-signature $\hat{\sigma}$.
- $\mathsf{pVrfy}_{vk}(m, Y, \hat{\sigma}) \rightarrow 0/1$: on input a verification key vk, a pre-signature $\hat{\sigma}$, an instance Y and a message $m \in \{0,1\}^*$, outputs a bit $b \in \{0,1\}$.
- $\mathsf{Adapt}(\hat{\sigma}, y) \rightarrow \sigma$: on input a pre-signature $\hat{\sigma}$ and a witness y, outputs a signature σ.
- $\mathsf{Ext}(\sigma, \hat{\sigma}, Y) \rightarrow y$: on input a signature σ, a pre-signature $\hat{\sigma}$ and an instance Y, outputs a witness y such that $(Y, y) \in \mathsf{R}$, or $\perp$.

In addition to the standard signature correctness, an AS has to satisfy pre-signature correctness. Informally, it guarantees that an honestly generated pre-signature w.r.t. an instance $Y \in \mathsf{R}$ is a valid pre-signature and can be completed into a valid signature from which a witness for Y can be extracted.

Definition 2 (Pre-signature correctness). *An adaptor signature scheme $\Pi_{\mathsf{R},\sum}$ satisfies pre-signature correctness if for every λ, every message $m \in \{0,1\}^*$ and every statement/witness pair $(Y, y) \in \mathsf{R}$, the following holds:*

$$\Pr\left[\begin{array}{c|c} \mathsf{pVrfy}_{vk}(m, Y, \hat{\sigma}) \rightarrow 1 & \mathsf{Gen}(1^\lambda) \rightarrow (sk, vk) \\ \wedge & \mathsf{pSign}_{sk}(m, Y) \rightarrow \hat{\sigma} \\ \mathsf{Vrfy}_{vk}(m, \sigma) \rightarrow 1 & \mathsf{Adapt}(\hat{\sigma}, y) \rightarrow \sigma \\ \wedge & \mathsf{Ext}(\sigma, \hat{\sigma}, Y) \rightarrow y' \\ (Y, y') \in \mathsf{R} & \end{array}\right] = 1$$

We now define the security properties of an AS scheme. We begin with the notion of unforgeability which is similar to the definition of existential unforgeability under chosen message attacks but additionally requires that producing a forgery σ for some message m is hard even given a pre-signature on m w.r.t. an instance $Y \in \mathsf{R}$. Let us emphasize that allowing the adversary to learn a pre-signature on the forgery message m is crucial since for our applications unforgeability needs to hold even in case the adversary learns a pre-signature for m without knowing a corresponding witness for Y. We formally define the existential unforgeability under chosen message attack for AS (aEUF–CMA).

Definition 3 (aEUF–CMA security). *An adaptor signature scheme $\Pi_{\mathsf{R},\sum}$ is aEUF–CMA secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that:* $\Pr[\mathit{aSigForge}_{\mathcal{A},\Pi_{\mathsf{R},\sum}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where the experiment $\mathit{aSigForge}_{\mathcal{A},\Pi_{\mathsf{R},\sum}}$ is defined as follows:*

```
$\mathrm{aSigForge}_{\mathcal{A},\Pi_{\mathsf{R},\sum}}(\lambda)$
$\mathcal{Q} = \emptyset$
$(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
$m \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Sign}_{sk}}(\cdot), \mathcal{O}_{\mathrm{pSign}_{sk}}(\cdot)}(vk)$
$(Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$
$\hat{\sigma} \leftarrow \mathsf{pSign}_{sk}(m, Y)$
$\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Sign}_{sk}}(\cdot), \mathcal{O}_{\mathrm{pSign}_{sk}}(\cdot)}(\hat{\sigma}, Y)$
return $(m \notin \mathcal{Q} \wedge \mathsf{Vrfy}_{vk}(m, \sigma))$

$\mathcal{O}_{\mathrm{Sign}_{sk}}(m)$
$\sigma \leftarrow \mathsf{Sign}_{sk}(m)$
$\mathcal{Q} = \mathcal{Q} \cup \{m\}$
return $\sigma$

$\mathcal{O}_{\mathrm{pSign}_{sk}}(m, Y)$
$\hat{\sigma} \leftarrow \mathsf{pSign}_{sk}(m, Y)$
$\mathcal{Q} = \mathcal{Q} \cup \{m\}$
return $\hat{\sigma}$
```

As discussed above, AS guarantees that a valid pre-signature w.r.t. Y can be completed to a valid signature if and only if the corresponding witness y for Y is known. An additional property that we will require is that any valid pre-signature w.r.t. Y (possibly produced by a malicious signer) can be completed into a valid signature using the witness y with $(Y, y) \in \mathsf{R}$. Notice that this property is stronger than the pre-signature correctness property, since we require that even maliciously produced pre-signatures can always be completed into valid signatures. The next definition formalizes the above discussion.

Definition 4 (Pre-signature adaptability). *An adaptor signature scheme $\Pi_{\mathsf{R},\sum}$ satisfies pre-signature adaptability if for any λ, any message $m \in \{0,1\}^*$, any statement/witness pair $(Y, y) \in \mathsf{R}$, any key pair $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and any pre-signature $\hat{\sigma}$ with $\mathsf{pVrfy}_{vk}(m, Y, \hat{\sigma}) \to 1$, we have $\Pr[\mathsf{Vrfy}_{vk}(m, \mathsf{Adapt}(\hat{\sigma}, y)) \to 1] = 1$.*

The aEUF–CMA security together with the pre-signature adaptability ensures that a pre-signature for Y can be transferred into a valid signature if and only if the corresponding witness y is known. The last property that we are interested in is witness extractability. Informally, it guarantees that a valid signature/pre-signature pair $(\sigma, \hat{\sigma})$ for message/statement (m, Y) can be used to extract the corresponding witness y.

Definition 5 (Witness extractability). *An adaptor signature scheme $\Pi_{\mathsf{R},\sum}$ is witness extractable if for every PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that the following holds:*

$$\Pr[aWitExt_{\mathcal{A},\Pi_{\mathsf{R},\sum}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where the experiment $aWitExt_{\mathcal{A},\Pi_{\mathsf{R},\sum}}$ is defined as follows

Let us stress that while the witness extractability experiment aWitExt looks fairly similar to the experiment aSigForge, there is one crucial difference; namely, the adversary is allowed to choose the forgery instance Y. Hence, we can assume that he knows a witness for Y so he can generate a valid signature on the forgery message m. However, this is not sufficient to win the experiment. The adversary wins only if the valid signature does not reveal a witness for Y.

```
$\mathrm{aWitExt}_{\mathcal{A},\Pi_{\mathsf{R},\sum}}(\lambda)$
$\mathcal{Q} = \emptyset$
$(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
$(m, Y) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Sign}_{sk}}(\cdot),\mathcal{O}_{\mathrm{pSign}_{sk}}(\cdot)}(vk)$
$\hat{\sigma} \leftarrow \mathsf{pSign}_{sk}(m, Y)$
$\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{Sign}_{sk}}(\cdot),\mathcal{O}_{\mathrm{pSign}_{sk}}(\cdot)}(\hat{\sigma})$
$y' \leftarrow \mathsf{Ext}(\sigma, \hat{\sigma}, Y)$
return $(m \notin \mathcal{Q} \wedge (Y, y') \notin \mathsf{R}$
$\wedge \mathsf{Vrfy}_{vk}(m, \sigma))$
```

```
$\mathcal{O}_{\mathrm{Sign}_{sk}}(m)$
$\sigma \leftarrow \mathsf{Sign}_{sk}(m)$
$\mathcal{Q} = \mathcal{Q} \cup \{m\}$
return $\sigma$
```

```
$\mathcal{O}_{\mathrm{pSign}_{sk}}(m, Y)$
$\hat{\sigma} \leftarrow \mathsf{pSign}_{sk}(m, Y)$
$\mathcal{Q} = \mathcal{Q} \cup \{m\}$
return $\hat{\sigma}$
```

3.2 ECDSA

We review the ECDSA scheme [15] $\sum_{\mathrm{ECDSA}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ on a message $m \in \{0,1\}^*$ as follows. Let $\mathbb{G}$ be an Elliptic curve group of order q with base point (generator) G and let $pp = (\mathbb{G}, G, q)$ be the public parameter.

- $\mathsf{Gen}(pp) \rightarrow (Q, x)$: The key generation algorithm uniformly chooses a secret signing key $x \leftarrow \mathbb{Z}_q$, and calculates the verification key $Q = x \cdot G$, and outputs $(sk = x, vk = Q)$.
- $\mathsf{Sign}_{sk}(m) \rightarrow (r, s)$. The signing algorithm chooses $k \leftarrow \mathbb{Z}_q$ randomly and computes $r = f(kG)$ and $s = k^{-1}(h(m) + rx)$, where $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a hash function modeled as a random oracle and $f : \mathbb{G} \rightarrow \mathbb{Z}_q$ is defined as the projection to the x-coordinate.
- $\mathsf{Vrfy}_{vk}(m, \sigma) \rightarrow 0/1$. The verification algorithm computes $r' = f(s^{-1} \cdot (m' \cdot G + r \cdot Q))$. If $r = r' \bmod q$, outputs 1, otherwise, outputs 0.

According to the structure of ECDSA, for same message m, if (r, s) is a valid signature, so is $(r, -s)$. $\sum_{\mathrm{ECDSA}}$ does not satisfy SUF-CMA security which we need to prove the security of ECDSA-AS. Following [18,8,2], we also use the positive ECDSA scheme which guarantees that if (r, s) is a valid signature, then $|s| \leq (q-1)/2$.

4 ECDSA-based Adaptor Signature

In this section, we present a construction of ECDSA-AS $\Pi_{\mathsf{R},\sum} = (\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Ext})$ w.r.t. a hard relation R and a ECDSA signature $\sum = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. Formally, we define hard relations $\mathsf{R} = \{(I_Y = (Y, \pi_Y), y) | Y = yG \wedge \mathsf{V}(I_Y) = 1\}$ and $\mathsf{R}_Z = \{(I_Z = (G, Q, Y, Z), x) | Q = xG \wedge Z = xY\}$, and denote by P the prover and by V the verifier of the proof system.

- $\mathsf{pSign}_{(vk,sk)}(m, I_Y) \rightarrow \hat{\sigma}$: on input a key-pair $(vk, sk) = (Q, x)$, a message m and an instance $I_Y = (Y, \pi_Y)$, the algorithm computes $Z = xY$, runs $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Q, Y, Z), x)$, and chooses $k \leftarrow \mathbb{Z}_q$, computes $r = f(kY)$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$ and outputs $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$.

– $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \to 0/1$: on input the verification key $vk = Q$, a message m, an instance I_Y, and a pre-signature value $\hat{\sigma}$, the algorithm outputs $\perp$ if $\mathsf{V}(I_Z) \to 0$, otherwise, it computes $r' = f(\hat{s}^{-1} \cdot (h(m) \cdot Y + r \cdot Z))$, and if $r' = r$, outputs 1, else outputs 0.
– $\mathsf{Adapt}(y, \hat{\sigma}) \to \sigma$: on input the witness y, and pre-signature $\hat{\sigma}$, the algorithm computes $s = \hat{s} \cdot y^{-1} \bmod q$ and outputs the signature $\sigma = (r, s)$.
– $\mathsf{Ext}(\sigma, \hat{\sigma}, I_Y) \to y$: on input the signature σ, the pre-signature $\hat{\sigma}$ and the instance I_Y, the algorithm computes $y = \hat{s}/s \bmod q$. If $(I_Y, y) \in \mathsf{R}$, it outputs y, else outputs $\perp$.

```
$\mathsf{pSign}_{(vk,sk)}(m, I_Y)$
$(vk, sk) = (Q, x), Z = xY$
$\mathsf{P}(I_Z, x) \to \pi_Z$
$k \leftarrow \mathbb{Z}_q$
$r = f(kY)$
$\hat{s} = k^{-1}(h(m) + rx) \bmod q$
return $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$
```

```
$\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma})$
if $\mathsf{V}(I_Z) \to 0$
outputs $\perp$.
else, $r' = f(\hat{s}^{-1} \cdot h(m) \cdot Y + \hat{s}^{-1} \cdot r \cdot Z))$
If $r' = r$, output 1
else, output 0.
return 0/1
```

```
$\mathsf{Adapt}(y, \hat{\sigma})$
$\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$
$s = \hat{s} \cdot y^{-1} \bmod q$
return $\sigma = (r, s)$
```

```
$\mathsf{Ext}(\sigma, \hat{\sigma}, I_Y) \to y$
$\sigma = (r, s), \hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$
$y = \hat{s}/s \bmod q$
If $(I_Y, y) \in \mathsf{R}$, it returns $y$,
else, it returns $\perp$
```

Fig. 1: ECDSA-based adaptor signature scheme

Note that in the pre-signing phase, our ECDSA-AS uses the pre-signing key x as the witness to prove the hard relation $((G, Q = xG, Y = yG, Z = xY), x)$ satisfies equality of discrete logarithms, while in [2], they use the random number k as the witness to prove the hard relation $((G, K = kG, Y = yG, \hat{K} = kY), k)$ satisfies equality of discrete logarithms. With this change, the zero-knowledge proof in our scheme can be executed offline, and in the online phase, the pre-signing algorithm is similar to the original ECDSA signing algorithm except for using (Z, Y) as verification key and base point instead of (Q, G). In addition, for the same hard relation (I_Y, y), our scheme only needs one zero-knowledge proof for pre-signing many different messages, but in [2], the zero-knowledge proof is linearly related to the number of pre-signatures.

Theorem 1. *Assuming that the positive ECDSA signature $\sum$ is SUF-CMA secure and R is a hard relation, the ECDSA-based adaptor signature $\Pi_{\mathsf{R},\sum}$ as defined in fig. 1 is secure in random oracle model.*

We prove that our ECDSA-AS scheme in fig. 1 satisfies pre-signature adaptability, pre-signature correctness, aEUF–CMA security, and witness extractability as follows.

Lemma 1. (Pre-signature adaptability) *The ECDSA-based adaptor signature scheme $\mathit{\Pi}_{\mathsf{R},\sum}$ satisfies pre-signature adaptability.*

Proof. For any $(I_Y, y) \in \mathsf{R}$, $m \in \{0,1\}^*$, $G, Q, Y, Z \in \mathbb{G}$ and $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$. For $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \rightarrow 1$. That is, $Y = yG, Z = xY = yQ = xyG$, $\hat{K} = (h(m) \cdot \hat{s}^{-1})Y + r \cdot \hat{s}^{-1}Z = kY$, $r' = f(\hat{K}) = f(kY) = r$.

By definition of Adapt, we know that $\mathsf{Adapt}(\hat{\sigma}, y) \rightarrow \sigma$, where $\sigma = (r, s), s = \hat{s} \cdot y^{-1} = (yk)^{-1}(h(m) + rx) \bmod q$. Hence, we have

$$K' = (h(m) \cdot s^{-1})G + r \cdot s^{-1}Q = kY.$$

Therefore, $r' = f(K') = f(kY) = r$. That is $\mathsf{Vrfy}_{vk}(m, \sigma) \rightarrow 1$.

Lemma 2. (Pre-signature correctness) *The ECDSA-based adaptor signature scheme $\mathit{\Pi}_{\mathsf{R},\sum}$ satisfies pre-signature correctness.*

Proof. For any $x, y \in \mathbb{Z}_q$, $Q = xG, Y = yG$ and $m \in \{0,1\}^*$. For $\mathsf{pSign}_{(vk,sk)}(m, I_Y) \rightarrow \hat{\sigma} = (r,\ \hat{s},\ Z,\ \pi_Z)$, it holds that $Y = yG, Z = xY$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$ for some $k \leftarrow \mathbb{Z}_q$. Set $\hat{K} = (h(m) \cdot \hat{s}^{-1})Y + r \cdot \hat{s}^{-1}Z = kY$.

Therefore, $r' = f(\hat{K}) = f(kY) = r$, we have $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \rightarrow 1$. By Lemma 1, this implies that $\mathsf{Vrfy}_{vk}(m, \sigma) \rightarrow 1$, for $\mathsf{Adapt}(\hat{\sigma}, y) \rightarrow \sigma = (r, s)$. By the definition of Adapt, we know that $s = \hat{s} \cdot y^{-1}$ and hence

$$\mathsf{Ext}(\sigma, \hat{\sigma}, I_Y) = \hat{s}/s = \hat{s}/(\hat{s}/y) = y.$$

Lemma 3. (aEUF–CMA security) *Assuming that the positive ECDSA signature scheme $\sum$ is SUF–CMA secure and R is a hard relation, the ECDSA-based adaptor signature scheme $\mathit{\Pi}_{\mathsf{R},\sum}$ as defined above is aEUF–CMA secure.*

Proof. We prove the aEUF–CMA security of the ECDSA-AS by reduction to strong unforgeability of positive ECDSA signatures. Our proof works by showing that, for any PPT adversary $\mathcal{A}$ breaking aEUF–CMA security of the ECDSA-AS, we can construct a PPT reduction algorithm $\mathcal{S}$ who can break the SUF–CMA security of ECDSA. $\mathcal{S}$ has access to the signing oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ of ECDSA and the random oracle $\mathcal{H}_{\text{ECDSA}}$, which it uses to simulate oracle queries for $\mathcal{A}$, namely random oracle ($\mathcal{H}$), signing oracle ($\mathcal{O}_{\text{Sign}}$) and pre-signing oracle ($\mathcal{O}_{\text{pSign}}$) queries.

The main challenge is simulating $\mathcal{O}_{\text{pSign}}$ queries since $\mathcal{S}$ can only get full signatures from its ECDSA signing oracle. Hence, $\mathcal{S}$ needs to transform full signatures into pre-signatures for $\mathcal{A}$. In order to do so, $\mathcal{S}$ needs to learn the witness y, for instance, I_Y, and simulate the zero-knowledge proof π_Z which proves $\exists\ x$ such that $Z = xY$ and $Q = xG$. More concretely, upon receiving a $\mathcal{O}_{\text{pSign}}$ query from $\mathcal{A}$ on input a message m and an instance I_Y, the simulator queries its signing oracle to obtain a full signature on m. Further, $\mathcal{S}$ learns the witness y, s.t. $Y = yG$, in order to transform the full signature into a pre-signature for $\mathcal{A}$. We make use of the extractability property of the zero-knowledge proof π_Y, in order to extract y and consequently transform a full signature into a valid pre-signature. What's more, we use the zero-knowledge property of proving π_Z, which allows for the simulation of the proof for a statement without knowing the corresponding witness.

We prove security by describing a sequence of games $G_0, \cdots, G_4$, where G_0 is the original aSigForge game. Then we show that for all $i = 0, \cdots, 3$, G_i and G_{i+1} are indistinguishable.

- Game G_0: This game corresponds to the original aSigForge game, where the adversary $\mathcal{A}$ has to come up with a valid forgery for a message m, while having access to oracles $\mathcal{H}$, $\mathcal{O}_{\text{pSign}}$ and $\mathcal{O}_{\text{Sign}}$.
- Game G_1: This game works exactly as G_0 with the exception that upon the adversary outputting a forgery σ^*, the game checks if completing the pre-signature $\hat{\sigma}$ using the witness y results in σ^*. In that case, the game aborts.
- Game G_2: This game only applies changes to the $\mathcal{O}_{\text{pSign}}$ oracle as opposed to the previous game G_1. Namely, during the $\mathcal{O}_{\text{pSign}}$ queries, this game extracts a witness y by executing the algorithm K on inputs the instance I_Y, the proof π and the list of random oracle query $\mathcal{H}$. The game aborts if for the extracted witness y it does not hold that $(I_Y, y) \in \mathsf{R}$.
- Game G_3: This game extends the changes of the previous game G_2 to the $\mathcal{O}_{\text{pSign}}$ oracle by first creating a valid full signature σ by executing the Sign algorithm and then converting σ into a pre-signature using the extracted witness y. Further, the game calculates $Z = yQ = xY$, and simulates a zero-knowledge proof π_S.
- Game G_4: In this game, upon receiving the challenge message m^* from $\mathcal{A}$, the game creates a full signature by executing the Sign algorithm and transforms the resulting signature into a pre-signature in the same way as in the previous game G_3 during the $\mathcal{O}_{\text{pSign}}$ execution.

There exists a simulator that perfectly simulates G_4 and uses $\mathcal{A}$ to win a positive ECDSA strongSigForge game. $\mathcal{S}$ simulates $\mathcal{A}$'s oracle queries as follows:

- Signing oracle queries: Upon $\mathcal{A}$ querying the oracle $\mathcal{O}_{Sign}$ on input m, $\mathcal{S}$ forwards m to its oracle $\mathcal{O}_{\text{ECDSA-sign}}$ and forwards its response to $\mathcal{A}$.
- Random oracle queries: Upon $\mathcal{A}$ querying the oracle $\mathcal{H}$ on input x, if $H[x] = \perp$, then $\mathcal{S}$ queries $\mathcal{H}_{\text{ECDSA}}(x)$, otherwise the simulator returns $\mathcal{H}[x]$.
- Pre-signing oracle queries:
  1) Upon $\mathcal{A}$ querying the oracle $\mathcal{O}_{\text{pSign}}$ on input (m, I_Y), the simulator extracts y using the extractability of NIZK_{I_Y}, forwards m to oracle $\mathcal{O}_{\text{ECDSA-sign}}$ and parses the signature as (r, s).
  2) $\mathcal{S}$ generates a pre-signature from (r, s) by computing $\hat{s} = s \cdot y$.
  3) $\mathcal{S}$ computes $Z = yQ = xY$ and simulates a zero-knowledge proof π_S, proving that $Q = xG$ and $Z = xY$ satisfy equality of discrete logarithms. The simulator outputs $(r, \hat{s}, Z, \pi_S)$.

In the challenge phase:

1) Upon $\mathcal{A}$ outputting the message m^* as the challenge message, $\mathcal{S}$ generates $(I_Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$, forwards m^* to the oracle $\mathcal{O}_{\text{ECDSA-sign}}$ and parses the signature as (r, s).
2) The simulator generates the required pre-signature $\hat{\sigma}$ in the same way as during $\mathcal{O}_{\text{pSign}}$ queries.
3) Upon $\mathcal{A}$ outputting a forgery σ^*, the simulator outputs (m^*, σ^*) as its own forgery.

We emphasize that the main difference between the simulation and G_4 are syntactical, namely, instead of generating the public and secret keys and running the algorithm Sign and the random oracle $\mathcal{H}$, the simulator $\mathcal{S}$ uses its oracles $\mathcal{O}_{\text{ECDSA-Sign}}$ and $\mathcal{H}$. It remains to show that the forgery output by $\mathcal{A}$ can be used by the simulator to win a positive ECDSA strongSigForge game.

Claim 1 *Let Bad_1 be the event that G_1 aborts, then* $\Pr[Bad_1] \leq \mathsf{negl}_1(\lambda)$.

Proof. We prove this claim using a reduction to the hardness of the relation R. More concretely, we construct a simulator $\mathcal{S}$ who can break the hardness of the hard relation assuming he has access to an adversary $\mathcal{A}$ that causes G_1 to abort with non-negligible probability. The simulator gets a challenge I_Y^*, upon which it generates a key pair $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ in order to simulate $\mathcal{A}$'s queries to the oracles $\mathcal{H}$, $\mathcal{O}_{\text{Sign}}$ and $\mathcal{O}_{\text{pSign}}$. This simulation of the oracles works as described in G_1. Eventually, upon receiving challenge message m from $\mathcal{A}$, $\mathcal{S}$ computes a pre-signature $\hat{\sigma} \leftarrow \mathsf{pSign}_{(vk,sk)}(m, I_Y^*)$, returns σ^* to $\mathcal{A}$ who outputs a forgery σ.

Assuming that Bad_1 happened (i.e. $\mathsf{Adapt}(\hat{\sigma}, y) = \sigma$), we know that due to the correctness property, the simulator can extract $y^* \leftarrow \mathsf{Ext}(\sigma, \sigma^*, I_Y^*)$ to obtain a valid statement/witness pair for the relation R, i.e.$(I_Y^*, y^*) \in \mathsf{R}$. First, we note that the view of $\mathcal{A}$ is indistinguishable from his view in G_1, since the challenge I_Y^* is an instance of the hard relation R and hence equally distributed to the public output of GenR. Hence the probability of $\mathcal{S}$ breaking the hardness of the relation is equal to the probability of the Bad_1 event. By our assumption, this is non-negligible which is the contradiction with the hardness of hard relation R.

Since games G_1 and G_0 are equivalent except if event Bad_1 occurs, it holds that $|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \leq \mathsf{negl}_1(\lambda)$.

Claim 2 *Let Bad_2 be the event that G_2 aborts during an $\mathcal{O}_{pSign}$ execution, then it holds that* $\Pr[Bad_2] \leq \mathsf{negl}_2(\lambda)$.

Proof. According to the online extractor property of the zero-knowledge proof, for a witness y extracted from a proof π of instance I_Y such that $\mathsf{V}(I_Y, \pi) \rightarrow 1$, it holds that $(I_Y, y) \in \mathsf{R}$ except with negligible probability in the security parameter. Since games G_2 and G_1 are equivalent except if event Bad_2 occurs, it holds that $|\Pr[G_2 = 1] - \Pr[G_1 = 1]| \leq \mathsf{negl}_2(\lambda)$.

Claim 3 *G_3 is computationally indistinguishable from the previous game G_2.*

Proof. This game extends the changes of the previous game to the $\mathcal{O}_{\text{pSign}}$ oracle by first creating a valid full signature σ by executing the Sign algorithm and then converting σ into a pre-signature using the extracted witness y. Further, the game calculates $Z = yQ = xY = xyG$ and simulates a zero-knowledge proof π_S.

Due to the zero-knowledge property of the zero-knowledge proof, the simulator can produce a proof π_S which is computationally indistinguishable from a proof $\pi_Z \leftarrow \mathsf{P}((G, Q, Y, Z), x)$. Hence, this game is indistinguishable from the previous game and it holds that $\Pr[G_3 = 1] \leq \Pr[G_2 = 1] + \mathsf{negl}_3(\lambda)$.

Claim 4 *G_4 is computationally indistinguishable from the previous game G_3.*

Proof. This proof follows above proof. Hence, G_4 is indistinguishable from G_3 and it holds that $\Pr[G_4 = 1] \leq \Pr[G_3 = 1] + \mathsf{negl}_4(\lambda)$.

Claim 5 *(m^*, σ^*) constitutes a valid forgery in positive ECDSA strongSigForge game.*

Proof. In order to prove this claim, we have to show that the tuple (m^*, σ^*) has not been output by the oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ before. Note that the adversary $\mathcal{A}$ has not previously made a query on the challenge message m^* to either $\mathcal{O}_{\text{Sign}}$ or $\mathcal{O}_{\text{pSign}}$. Hence, $\mathcal{O}_{\text{ECDSA-Sign}}$ is only queried on m^* during the challenge phase. As shown in game G_1, the adversary outputs a forgery σ^* which is equal to the signature σ output by $\mathcal{O}_{\text{ECDSA-Sign}}$ during the challenge phase only with negligible probability.

Hence, $\mathcal{O}_{\text{ECDSA-Sign}}$ has never output σ^* on query m^* before and consequently (m^*, σ^*) constitutes a valid forgery for positive ECDSA strongSigForge game.

From the games G_0 to G_4, we get that $|\Pr[G_0 = 1] - \Pr[G_4 = 1]| \leq \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) + \mathsf{negl}_3(\lambda) + \mathsf{negl}_4(\lambda) \leq \mathsf{negl}(\lambda)$. Since $\mathcal{S}$ provides a perfect simulation of game G_4, we obtain:

$$\begin{aligned}\Pr[\text{aSigForge}_{\mathcal{A},\Pi_{\mathsf{R},\sum}}(\lambda) = 1] = \Pr[G_0 = 1] &\leq \Pr[G_4 = 1] + \mathsf{negl}(\lambda) \\ &\leq \Pr[\text{sSigForge}_{\mathcal{A},\sum}(\lambda) = 1] + \mathsf{negl}(\lambda).\end{aligned}$$

Lemma 4. (Witness extractability). *Assuming that the ECDSA scheme is SUF–CMA-secure and* R *is a hard relation, the ECDSA-based adaptor signature scheme $\Pi_{\mathsf{R},\sum}$ as defined is witness extractable.*

Proof. Our proof is to reduce the witness extractability of $\Pi_{\mathsf{R},\sum}$ to the strong unforgeability of the positive ECDSA. In other words, assuming that there exists a PPT adversary $\mathcal{A}$ who wins the aWitExt experiment, we can construct a PPT reduction algorithm $\mathcal{S}$ that wins positive ECDSA strongSigForge experiment.

During the reduction, the main challenge is to simulate a pre-signing oracle. Unlike in the aSigForge experiment, in aWitExt experiment, $\mathcal{A}$ outputs the instance I_Y for relation R alongside the challenge message m^*, meaning that $\mathcal{S}$ does not choose the pair (I_Y, y). Fortunately, it is possible to extract y from the zero-knowledge proof embedded in I_Y. After extracting y, the same approach used in order to simulate the pre-signing oracle queries can be taken here as well.

We prove security by first describing a sequence of games $G_0, \cdots, G_4$, where G_0 is the original aWitExt game. Then we show that for all $i = 0, \cdots, 3$, G_i and G_{i+1} are indistinguishable.

- Game G_0: This game corresponds to the original aWitExt game, where the adversary $\mathcal{A}$ has to come up with a valid signature σ for a message m, a given pre-signature $\hat{\sigma}$ and a given statement/witness pair (I_Y, y), while having access to oracles $\mathcal{H}$, $\mathcal{O}_{\text{pSign}}$ and $\mathcal{O}_{\text{Sign}}$, such that $(I_Y, \mathsf{Ext}(\sigma, \hat{\sigma}, I_Y)) \notin \mathsf{R}$.
- Game G_1: This game only applies changes to the $\mathcal{O}_{\text{pSign}}$ oracle as opposed to the previous game G_0. Namely, during the $\mathcal{O}_{\text{pSign}}$ queries, this game extracts a witness y by executing the algorithm K on inputs the instance I_Y, the proof π_Y and the list of random oracle query $\mathcal{H}$. The game aborts if for the extracted witness y it does not hold that $(I_Y, y) \in \mathsf{R}$.
- Game G_2: This game extends the changes to $\mathcal{O}_{\text{pSign}}$ from the previous game G_1. In the $\mathcal{O}_{\text{pSign}}$ execution, this game first creates a valid full signature σ by executing the Sign algorithm and converts σ into a pre-signature using the extracted witness y. Further, $\mathcal{S}$ calculates $Z = yQ$ and simulates a zero-knowledge proof π_S.
- Game G_3: In this game, we apply the same changes made in the game G_1 in oracle $\mathcal{O}_{\text{pSign}}$ to the challenge phase of the game. During the challenge phase, this game extracts a witness y by executing the algorithm K on inputs the instance I_Y, the proof π, and the list of random oracle queries $\mathcal{H}$. The game aborts if for the extracted witness y it does not hold that $(I_Y, y) \in \mathsf{R}$.
- Game G_4: In this game, we apply the same changes made in the game G_2 in oracle $\mathcal{O}_{\text{pSign}}$ to the challenge phase of the game. In the challenge phase, this game first creates a valid full signature σ by executing the Sign algorithm and converts σ into a pre-signature using the extracted witness y. Further, $\mathcal{S}$ calculates $Z = yQ$ and simulates a zero-knowledge proof π_S.

Having shown that the transition from the original aWitExt game (Game G_0) to Game G_4 is indistinguishable, it remains to show that there exists a simulator that perfectly simulates G_4 and uses $\mathcal{A}$ to win positive ECDSA strongSigForge game. $\mathcal{S}$ simulates $\mathcal{A}$'s oracle queries as follows:

- Signing oracle queries: Upon $\mathcal{A}$ querying the oracle $\mathcal{O}_{\text{Sign}}$ on input m, $\mathcal{S}$ forwards m to its oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ and forwards its response to $\mathcal{A}$.
- Random oracle queries: Upon $\mathcal{A}$ querying the oracle $\mathcal{H}$ on input x, if $\mathcal{H}[x] = \perp$, then $\mathcal{S}$ queries $\mathcal{H}_{\text{ECDSA}}(x)$, otherwise the simulator returns $\mathcal{H}[x]$.
- Pre-signing oracle queries:
  1) Upon $\mathcal{A}$ querying the oracle $\mathcal{O}_{\text{pSign}}$ on input (m, I_Y), the simulator extracts y using the extractability of NIZK_{I_Y}, forwards m to oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ and parses the signature as (r, s).
  2) $\mathcal{S}$ generates a pre-signature from (r, s) by computing $\hat{s} = s \cdot y$.
  3) $\mathcal{S}$ computes $Z = yQ = xY$ and simulates a zero-knowledge proof π_S, proving that $Q = xG$ and $Z = xY$ satisfy equality of discrete logarithms. The simulator outputs $(r, \hat{s}, Z, \pi_S)$.

In the challenge phase:

1) Upon $\mathcal{A}$ outputting the message (m^*, I_Y) as the challenge message, $\mathcal{S}$ extracts y using the extractability of NIZK_{I_Y}, forwards m^* to the oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ and parses the signature as (r, s).
2) The simulator generates the required pre-signature $\hat{\sigma}$ in the same way as during $\mathcal{O}_{\text{pSign}}$ queries.
3) Upon $\mathcal{A}$ outputting a forgery σ, the simulator outputs (m^*, σ^*) as its own forgery.

We emphasize that the main difference between the simulation and G_4 are syntactical, namely, instead of generating the public and secret keys and running the algorithm Sign and the random oracle $\mathcal{H}$, the simulator $\mathcal{S}$ uses its oracles $\mathcal{O}_{\text{ECDSA-Sign}}$ and $\mathcal{H}_{\text{ECDSA}}$. It remains to show that the signature output by $\mathcal{A}$ can be used by the simulator to win a positive ECDSA strongSigForge game.

Claim 6 *Let Bad_1 be the event that G_1 aborts during an $\mathcal{O}_{pSign}$ execution, then it holds that* $\Pr[Bad_1] \leq \mathsf{negl}_1(\lambda)$.

Proof. According to the online extractor property of the zero-knowledge proof, for a witness y extracted from a proof π for instance I_Y such that $\mathsf{V}(I_Y) = 1$, it holds that $(I_Y, y) \in \mathsf{R}$ except with negligible probability. Since games G_1 and G_0 are equivalent except if event Bad_1 occurs, it holds that $|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \leq \mathsf{negl}_1(\lambda)$.

Claim 7 *Let Bad_2 be the event that G_3 aborts during the challenge phase, then it holds that* $\Pr[Bad_2] \leq \mathsf{negl}_3(\lambda)$.

Proof. This proof is analogous to above proof. Since games G_2 and G_3 are equivalent except if event Bad_2 occurs, it holds that $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \mathsf{negl}_3(\lambda)$.

Claim 8 *G_2 is computationally indistinguishable from the previous game G_1.*

Proof. This game extends the changes to $\mathcal{O}_{\text{pSign}}$ from the previous game. In the $\mathcal{O}_{\text{pSign}}$ execution, this game first creates a valid full signature σ by executing the Sign algorithm and converts σ into a pre-signature using the extracted witness y. Further, the game calculates $Z = yQ = xY$ and simulates a zero-knowledge proof π_S. Due to the zero-knowledge property of the zero-knowledge proof, the simulator can produce a proof $\pi_S \leftarrow \mathsf{S}((G, Q, Y, Z), 1)$ which is indistinguishable from a proof $\pi_Z \leftarrow \mathsf{P}((G, Q, Y, Z), x)$. Hence, G_2 is computationally indistinguishable from the previous game G_1. It holds that $\Pr[G_1 = 1] - \Pr[G_2 = 1] \leq \mathsf{negl}_2(\lambda)$.

Claim 9 *G_4 is computationally indistinguishable from the previous game G_3.*

Proof. This proof is analogous to above proof. G_4 is computationally indistinguishable from the previous game G_3, that is, it holds that $|\Pr[G_4 = 1] - \Pr[G_3 = 1]| \leq \mathsf{negl}_4(\lambda)$.

Claim 10 (m^*, σ^*) *constitutes a valid forgery in positive ECDSA strongSigForge game.*

Proof. In order to prove this claim, we have to show that the tuple (m^*, σ^*) has not been output by the oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ before. Note that the adversary A has not previously made a query on the challenge message m^* to either $\mathcal{O}_{\text{pSign}}$ or $\mathcal{O}_{\text{Sign}}$. Hence, $\mathcal{O}_{\text{ECDSA-Sign}}$ is only queried on m^* during the challenge phase. If the adversary outputs a forgery σ^* which is equal to the signature σ output by $\mathcal{O}_{\text{ECDSA-Sign}}$ during the challenge phase, the extracted y would be in relation with the given public value I_Y. Hence, $\mathcal{O}_{\text{ECDSA-Sign}}$ has never output σ^* on query m^* before and consequently (m^*, σ^*) constitutes a valid forgery for positive ECDSA strongSigForge game.

From the games G_0 to G_4 we get that $|\Pr[G_0 = 1] - \Pr[G_4 = 1]| \leq \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) + \mathsf{negl}_3(\lambda) + \mathsf{negl}_4(\lambda) \leq \mathsf{negl}(\lambda)$. Since $\mathcal{S}$ provides a perfect simulation of game G_4, we obtain:

$$\begin{aligned}\Pr[\text{aWitExt}_{\mathcal{A},\Pi_{\mathsf{R},\Sigma}}(\lambda) = 1] = \Pr[G_0 = 1] &\leq \Pr[G_4 = 1] + \mathsf{negl}(\lambda) \\ &\leq \Pr[\text{sSigForge}_{\mathcal{A},\Sigma}(\lambda) = 1] + \mathsf{negl}(\lambda).\end{aligned}$$

5 Fast ECDSA-based Adaptor Signature with Offline Zero-Knowledge Proof

In this section, according to the structure $Z = xY = yQ$, we can use the "offline proof technique" to construct two fast ECDSA-AS schemes called ECDSA-AS1/2. For the structure $Z = xY$ in ECDSA-AS1, the prover only proves that there exists witness x, such that $Q = xG$ and $Z = xY$ satisfy equality of discrete logarithms. For the structure $Z = yQ$ in ECDSA-AS2, the prover only proves that there exists witness y, such that $Y = yG$ and $Z = yQ$ satisfy equality of discrete logarithms.

In our ECDSA-AS, the witness of hard relation $\mathsf{R}_Z = ((G, Q, Y, Z), x)$ is the pre-signing key x, and the zero-knowledge proof is $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Q, Y, Z), x)$, so the signer can compute the zero-knowledge proof offline before getting the message. Therefore, ECDSA-AS1 can be designed from our ECDSA-AS directly with offline zero-knowledge proof. Refer to the section 4 for specific construction which is ignored here.

Because of the structure $Z = yQ$, by using y as the witness of hard relation $\mathsf{R}_Z = ((G, Y, Q, Z), y)$, and proving that there exists the witness y, such that $Y = yG$ and $Z = yQ$, that is, the zero-knowledge proof is $\pi_Z \leftarrow \mathsf{P}((G, Y, Q, Z), y)$, we can construct efficient ECDSA-AS2 as follows. Formally, we define hard relations $\mathsf{R} = \{(I_Z = (G, Y, Q, Z, \pi_Y, \pi_Z), y) | Y = yG \wedge Z = yQ \wedge \mathsf{V}(Y, \pi_Y) = 1 \wedge \mathsf{V}((G, Y, Q, Z), \pi_Z) = 1\}$.

- $\mathsf{pSign}_{(vk,sk)}(m, I_Z) \rightarrow \hat{\sigma}$: on input a key-pair $(vk, sk) = (Q, x)$, a message m and an instance $I_Z = (G, Y, Q, Z, \pi_Y)$, the algorithm runs $\mathsf{V}(I_Z) \rightarrow 0$, outputs $\perp$, otherwise, chooses $k \leftarrow \mathbb{Z}_q$, computes $r = f(kY)$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$ and outputs $\hat{\sigma} = (r, \hat{s})$.

- $\mathsf{pVrfy}_{vk}(m, I_Z, \hat{\sigma}) \rightarrow 0/1$: on input the verification key $vk = Q$, a message m, an instance I_Z, and a pre-signature value $\hat{\sigma}$, the algorithm outputs $\perp$ if $\mathsf{V}(I_Z) \rightarrow 0$, otherwise, it computes $r' = f(\hat{s}^{-1} \cdot (h(m) \cdot Y + r \cdot Z))$, and if $r' = r$, outputs 1, else outputs 0.
- $\mathsf{Adapt}(y, \hat{\sigma}) \rightarrow \sigma$: on input the witness y, and pre-signature $\hat{\sigma}$, the algorithm computes $s = \hat{s} \cdot y^{-1} \bmod q$ and outputs the signature $\sigma = (r, s)$.
- $\mathsf{Ext}(\sigma, \hat{\sigma}, I_Z) \rightarrow y$: on input the signature σ, the pre-signature $\hat{\sigma}$ and the instance I_Z, the algorithm computes $y = \hat{s}/s \bmod q$. If $(I_Z, y) \in \mathsf{R}$, it outputs y, else outputs $\perp$.

```
$\mathsf{pSign}_{(vk,sk)}(m, I_Z)$
if $\mathsf{V}(I_Z) \rightarrow 0$
output $\perp$.
else, $k \leftarrow \mathbb{Z}_q$
$r = f(kY)$
$\hat{s} = k^{-1}(h(m) + rx) \bmod q$
return $\hat{\sigma} = (r, \hat{s})$
```

```
$\mathsf{pVrfy}_{vk}(m, I_Z, \hat{\sigma})$
if $\mathsf{V}(I_Z) \rightarrow 0$
outputs $\perp$.
else, $r' = f(\hat{s}^{-1} \cdot h(m) \cdot Y + \hat{s}^{-1} \cdot r \cdot Z))$
If $r' = r$, output 1
else, output 0.
return 0/1
```

```
$\mathsf{Adapt}(y, \hat{\sigma})$
$\hat{\sigma} = (r, \hat{s})$
$s = \hat{s} \cdot y^{-1} \bmod q$
return $\sigma = (r, s)$
```

```
$\mathsf{Ext}(\sigma, \hat{\sigma}, I_Z) \rightarrow y$
$\sigma = (r, s), \hat{\sigma} = (r, \hat{s})$
$y = \hat{s}/s \bmod q$
If $(I_Z, y) \in \mathsf{R}$, it returns $y$,
else, it returns $\perp$
```

Fig. 2: ECDSA-based adaptor signature with offline proof

Note that the construction of ECDSA-AS1 is similar to our ECDSA-AS, except that the prover computes $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Q, Y, Z), x)$ offline. In the same way, ECDSA-AS2 is similar to ECDSA-AS2 except that the prover computes $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Y, Q, Z), y)$ offline.

Correctness. Following the lemma 1 and lemma 2, our ECDSA-AS1/2 schemes also satisfy pre-signature adaptability and pre-signature correctness.

Security. Our ECDSA-AS1/2 schemes embed the hard relation $(I_Y = (Y, \pi_Y), y)$ which satisfies the "self-proving structure" [2]. Therefore, in the security proof, the simulator can get the witness y and simulate the pre-signing oracle. Following the lemma 3 and lemma 4, our ECDSA-AS1/2 schemes also satisfy aEUF–CMA security and witness extractability.

Comparison. In [2], the prover computes proof $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, K, Y, \hat{K}), k)$ with the witness k which is the random number used in pre-signature. In ECDSA-AS1, the prover computes proof $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Q, Y, Z), x)$ with the witness

x which is the pre-signing key. In ECDSA-AS2, the prover computes proof $\pi_Z \leftarrow \mathsf{P}(I_Z = (G, Y, Q, Z), y)$ with the witness y which is also the witness of the hard relation $(I_Y = (Y, \Pi_Y), y)$.

As we can see, both zero-knowledge proofs of ECDSA-AS1/2 are independent of the message and random number used in the pre-signing algorithm, so this part can be run offline to improve the efficiency of the online pre-signing phase. At the same time, for embedding the same hard relation $(I_Y = (Y, \pi_Y), y)$, one zero-knowledge proof for $((G, Q, Y, Z), x)$ or $((G, Y, Q, Z), y)$ can be used to pre-signing many messages, but due to using the random number k as the witness in [2], each pre-signature needs new random number, the size of zero-knowledge proof is linearly related to the number of pre-signatures.

What's more, the structure of pre-signature in ECDSA-AS1/2 is similar to the original ECDSA except that the signer uses $Y = yG$ and $Z = yQ$ instead of G and $Q = xG$. Therefore, ECDSA-AS1/2 can reuse the hardware and software implementation of the ECDSA signing algorithm.

Furthermore, in ECDSA-AS1, the witness is pre-signing key x, so the prover must be the signer like [2], in which the witness is random number k. In ECDSA-AS2, the witness is y, so the prover can be the hard relation chooser who chooses the hard relation $(I_Y = (Y, \pi_Y), y)$. In order to construct two-party or threshold ECDSA-AS, many parties need to share the pre-signing key x and the random number k, so the zero-knowledge proof of proving the equality of discrete logarithms for $((G, Q, Y, Z), x)$ or $((G, K, Y, \hat{K}), k)$ should be distributed, which may be difficult to achieve. However, in ECDSA-AS2, the witness of $((G, Y, Q, Z), y)$ is y, which need not be shared, so zero-knowledge proof can be computed easily. Therefore, our ECDSA-AS2 is easy to extend to distributed scenarios, following the extensions of existing ECDSA, such as two-party and threshold mode, etc.

6 Performance and Experimental Results

6.1 Theoretical Analysis

As is shown in Table 1, we give the theoretical analysis of communication cost and efficiency about ECDSA-AS [14,2] and our ECDSA-AS schemes, respectively. For convenience, we omit some simple operations of modular multiplication and addition. The first ECDSA-AS proposed by Moreno-Sanchez et al. [14] does not provide provable security. Then Aumayr et al. [2] uses self-proving structure $\{(I_Y = (Y, \pi_Y), y) | Y = yG \wedge \mathsf{V}(Y, \pi_Y) \rightarrow 1\}$, and gives a provably secure ECDSA-AS based on [14]. But this scheme requires that proving $\hat{K} = kG$ and $K = kY$ satisfy equality of discrete logarithms with the witness k that is the random number in the pre-signing phase. For each message to be signed, the signer needs to choose a new random number and needs to compute a new zero-knowledge proof. At the same time, in order to transform ECDSA-AS into the two-party mode, the random number needs to be shared that leads to more complicated zero-knowledge proof.

Our ECDSA-AS schemes also use self-proving structure $\{(I_Y = (Y, \pi_Y), y) | Y = yG \wedge \mathsf{V}(Y, \pi_Y) \rightarrow 1\}$ to realize provable security. Meanwhile, in the pre-signing phase, our schemes use the witness x or y to prove $Z = xY$ and $Q = xG$ satisfy equality of discrete logarithms, or $Z = yQ$ and $Y = yG$ satisfy equality of discrete logarithms. Both proofs are independent of the message and random number, so our schemes can use "offline proof" to improve the efficiency. ECDSA-AS1/2 only computes once point multiplication operation, while ECDSA-AS in [14] and [2] need four times point multiplication operation. Benefiting from holding the ECDSA signing structure, our schemes can enjoy the same efficiency and completely reuse the hardware and software implementation of ECDSA. What's more, for embedding the same hard relation (I_Y, y), one zero-knowledge proof π_Z for $((G, Q, Y, Z), x)$ or $((G, Y, Q, Z), y)$ can be used to pre-signing many messages, while the size of zero-knowledge proof [14,2] is linearly related to the number of pre-signatures.

Table 1: Communication Cost and Efficiency Comparison

Schemes	pk size	sk size	signature size	online pSign	pVrfy	provable security
ECDSA-AS	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_q\rvert$	$\lvert\mathbb{G}\rvert + 4\lvert\mathbb{Z}_q\rvert$	4PM	6PM	?
ECDSA-AS	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_q\rvert$	$\lvert\mathbb{G}\rvert + 4\lvert\mathbb{Z}_q\rvert$	4PM	6PM	$\surd$
Our ECDSA-AS	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_q\rvert$	$\lvert\mathbb{G}\rvert + 4\lvert\mathbb{Z}_q\rvert$	4PM	6PM	$\surd$
Our ECDSA-AS2	$\lvert\mathbb{G}\rvert$	$\lvert\mathbb{Z}_q\rvert$	$2\lvert\mathbb{Z}_q\rvert$	PM	6PM	$\surd$

‡ $|\mathbb{G}|$ and $|\mathbb{Z}_q|$ denotes the size of the element in the group $\mathbb{G}$ and $\mathbb{Z}_q$, respectively. PM denotes point multiplication operation.

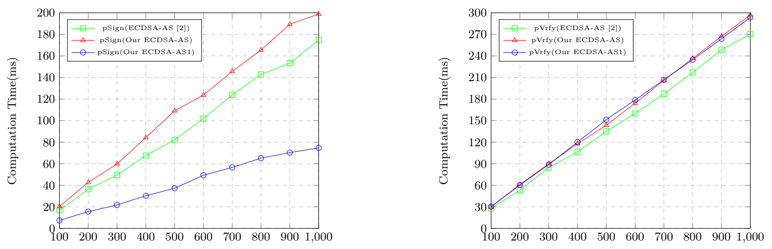

Fig. 3: Efficiency comparison of pre-signing and pre-verification operation

6.2 Experimental Analysis

In order to evaluate the practical performance of our schemes, we implement the ECDSA-AS [2], and our ECDSA-AS and ECDSA-AS1 based on the OpenSSL library. The program is executed on an Intel Core i5 CPU 2.3 GHz and 8GB RAM running macOS High Sierra 10.13.3 system.

As depicted in Figure 3, we run our implementations on the standard NIST curves. We run ECDSA-AS [2], our ECDSA-AS and ECDSA-AS1 many times respectively, and show the efficiency of online pre-signing and a verification operation. Based on the experimental data and analysis results, the average running times of all algorithms in ECDSA-AS and ECDSA-AS1 are the level of microseconds. Especially, the average running times over 1000 executions of the online pre-signing operation in ECDSA-AS [2], our ECDSA-AS and ECDSA-AS1 are 174.75 μs, 198.91 μs and 74.74 μs. Therefore, our protocol can compare with the state-of-the-art ECDSA-AS [2].

7 Conclusions

In this paper, we propose an ECDSA-AS and give the security proof based on ECDSA in the random oracle model. Compared with existing ECDSA-AS [14,2], we can use the offline-proof technique to improve the efficiency of the online pre-signing algorithm and then construct two efficient ECDSA-AS called ECDSA-AS1 and ECDSA-AS2. For embedding the same hard relation $(I_Y, y) \in \mathsf{R}$, one proof π_Z of hard relation R_Z used in the pre-signing phase can be used to pre-signing many messages. What's more, the pre-signing algorithms of ECDSA-AS1/2 are similar to the original ECDSA signing algorithm except for modifying some parameters, they can completely reuse the hardware and software implementation of ECDSA and upgrade existing ECDSA applications friendly.

References

1. Poelstra, A.: Lightning in scriptless scripts. mimblewimble team mailing list (2017), `https://lists.launchpad.net/mimblewimble/msg00086.html`
2. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. IACR Cryptol. ePrint Arch. 2020, 476 (2020), `https://eprint.iacr.org/2020/476`
3. Bitcoin Wiki: Payment channels (2018), `https://en.bitcoin.it/wiki/Paymentchannels`
4. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings. pp. 3–18. Springer (2015)
5. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016), `https://lightning.network/lightning-network-paper.pdf`
6. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Bitcoin-compatible virtual channels. IACR Cryptol. ePrint Arch. 2020, 554 (2020), `https://eprint.iacr.org/2020/554`
7. Eckey, L., Faust, S., Hostáková, K., Roos, S.: Splitting payments locally while routing interdimensionally. IACR Cryptol. ePrint Arch. 2020, 555 (2020)
8. 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, San Diego, California, USA, February 24-27, 2019. The Internet Society (2019)

9. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11598, pp. 508–526. Springer (2019)
10. Esgin, M.F., Ersoy, O., Erkin, Z.: Post-quantum adaptor signatures and payment channel networks. In: Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12309, pp. 378–397. Springer (2020)
11. Deshpande, A., Herlihy, M.: Privacy-preserving cross-chain atomic swaps. In: Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12063, pp. 540–549. Springer (2020)
12. Gugger, J.: Bitcoin-monero cross-chain atomic swap. IACR Cryptol. ePrint Arch. 2020, 1126 (2020), https://eprint.iacr.org/2020/1126
13. Schnorr, C.: Efficient identification and signatures for smart cards. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 239–252 (1989)
14. Moreno-Sanchez, P., Kate, A.: Scriptless scripts with ecdsa. lightning-dev mailing list https://lists.linuxfoundation.org/pipermail/lightning-dev/attachments/20180426/fe978423/attachment-0001.pdf
15. American National Standards Institute: X9.62: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa) (2005)
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145. IEEE Computer Society (2001), https://doi.org/10.1109/SFCS.2001.959888
17. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3621, pp. 152–168. Springer (2005)
18. Lindell, Y.: Fast secure two-party ECDSA signing. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. pp. 613–644 (2017), https://doi.org/10.1007/978-3-319-63715-0_21