Efficient ECDSA-based Adaptor Signature for Batched Atomic Swaps

Binbin Tu[1,2,3][0000−0002−2167−0762], Min Zhang[1,2,3][0009−0002−4772−6565], and Yu Chen[1,2,3(✉)][0000−0003−2553−1281]

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao 266237, China
{tubinbin,zm_min}@mail.sdu.edu.cn, yuchen@sdu.edu.cn

Abstract. Adaptor signature is a novel cryptographic primitive which ties together the signature and the leakage of a secret value. It has become an important tool for solving the scalability and interoperability problems in the blockchain. Aumayr et al. (Asiacrypt 2021) recently provide the formalization of the adaptor signature and present a provably secure ECDSA-based adaptor signature, which requires zero-knowledge proof in the pre-signing phase to ensure the signer works correctly. However, the number of zero-knowledge proofs is linear with the number of participants.

In this paper, we propose efficient ECDSA-based adaptor signature schemes and give security proofs based on ECDSA. In our schemes, the zero-knowledge proofs in the pre-signing phase can be generated in a batch and offline. Meanwhile, the online pre-signing algorithm is similar to the ECDSA signing algorithm and can enjoy the same efficiency as ECDSA. In particular, considering specific verification scenarios, such as (batched) atomic swaps, our schemes can reduce the number of zero-knowledge proofs in the pre-signing phase to one, independent of the number of participants. Last, we conduct an experimental evaluation, demonstrating that the performance of our ECDSA-based adaptor signature reduces online pre-signing time by about 60% compared with the state-of-the-art ECDSA-based adaptor signature.

Keywords: Adaptor signature · ECDSA-based adaptor signature · Batched atomic swaps · Blockchain.

1 Introduction

Adaptor signatures (AS), also known as scriptless scripts, are introduced by Poelstra [19] and recently formalized by Aumayr et al. [2]. It can be seen as an extension over a digital signature with respect to leaking a secret to certain parties. Namely, the signer uses a signing key to compute a pre-signature of a

message and a statement of a hard relation (e.g., the discrete logarithm), such that the pre-signature can be adapted into a (full) signature by the witness of the hard relation. Meanwhile, the witness can be extracted from the pre-signature and the full signature. AS provides the following intuitive properties: (i) Only the user knowing the signing key can generate a pre-signature; (ii) Only the user knowing the witness of the hard relation can convert a pre-signature into a full signature; (iii) Anyone holding a pre-signature and corresponding full signature can extract the witness.

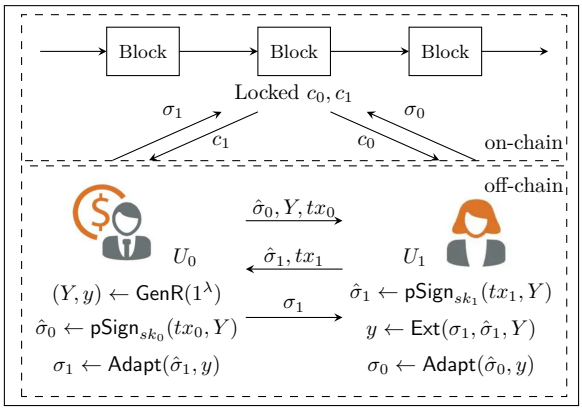

Fig. 1: The atomic swap protocol based on adaptor signature

To demonstrate the idea of AS, we introduce its key application atomic swaps in Figure 1. An atomic swap [10] can be defined between two users U_0 and U_1 who want to exchange two different cryptocurrencies c_0 and c_1. The crucial point of the exchange is ensuring fairness, i.e., both parties receive their expected output or nothing. Two parties U_0 and U_1 first set the time-lock for c_0 with the timeout t_0 and c_1 with the timeout t_1 on-chain[4]. Then, U_0 chooses a hard relation $(Y, y) \in R$ and pre-signing a transaction tx_0 for spending the coins c_0 to U_1, and then sends the pre-signature $\hat{\sigma}_0$, tx_0, Y to U_1. U_1 can check the validity of $\hat{\sigma}_0$ and pre-signing a transaction tx_1 for spending the coins c_1 to U_0 and then sends the pre-signature $\hat{\sigma}_1$, tx_1 to U_0. U_0 can check the validity of $\hat{\sigma}_1$ and adapts $\hat{\sigma}_1$ into the full signature σ_1 by the witness y, and then publishes σ_1 on the blockchain to get the coin c_1 within t_1. U_1 can extract the witness y from σ_1 and $\hat{\sigma}_1$ and adapts $\hat{\sigma}_0$ into σ_0, then publishes σ_0 on the blockchain to get the coin c_0 within t_0. As we can see, the pre-signature and the cryptographic condition need not to be published on-chain, compared with using the *Hash Time-Lock Contracts* (HTLCs) [20,15], AS reduces the operations on-chain and weaken scripting restrictions on the underlying blockchain.

[4] Both parties use time-lock to lock the exchange coins on-chain, and the timeouts $t_1 < t_0$ to ensure that U_1 can have enough time to react.

By tying the signing processing to the revelation of a secret value, AS brings about various advantages as follows: (i) Reducing the operations on-chain; (ii) Supporting advanced functionality beyond the limitation of the blockchains scripting language; (iii) Improving fungibility of transactions. To be specific, the pre-signature is generated and verified off-chain and only the full signature is published on-chain, so AS reduces the additional storage and verification costs greatly on-chain, meanwhile, it is not limited by the blockchains scripting language. Based on this advantage, Aumayr et al. [2] give a generalized channel construction by using AS as a key technique, which is compatible with any blockchain supporting transaction authorization, time-locks, and constant number of Boolean $\wedge$ and $\vee$ operations - requirements fulfilled by many (non-Turing-complete) blockchains including the Bitcoin. The fungibility property is said that the pre-signature embedded in the cryptographic condition (hard relations) inside is indistinguishable from a regular signature, and it can be used to hide payment channel network transactions among any other transactions [16]. Benefiting from above advantages, AS has also been shown highly useful in many blockchain applications such as payment channels [2,4,7,20,3], payment routing in payment channel networks [9,16,17,10], and atomic swaps [8,13,10].

Poelstra [19] first gives a Schnorr-based AS that is limited to cryptocurrencies using Schnorr signatures [21]. Moreno-Sanchez and Kate [18] present an ECDSA-based AS and its two-party version without provable security. Malavolta et al. [16] present two-party AS based on ECDSA [1], but 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 [5]. They exquisitely modify the hard relation in [18], by adding a zero-knowledge proof such that the witness can be extracted in the random oracle model [12]. For convenience, we name this modification as "self-proving structure". However, their ECDSA-based AS is not entirely satisfactory. In the pre-signing phase, the signer uses the random value as a witness to compute a pre-signing public parameter and a corresponding zero-knowledge proof. Especially, in the case of multiple participants, such as (batched) atomic swaps or multi-hop payments [16,10], the number of zero-knowledge proofs is *linear* with the number of participants. Therefore, we consider the following question in this work:

Is it possible to design an efficient ECDSA-based AS in which the number of zero-knowledge proofs in the pre-signing phase is independent of the number of participants?

1.1 Our Contributions

In this paper, we give an affirmative answer to the above question. First, we propose an ECDSA-based AS (ECDSA-AS) and prove the security based on

positive ECDSA in UC framework following [2]. Then, we develop more efficient ECDSA-AS schemes in which the zero-knowledge proofs in the pre-signing phase can be generated in a batch and offline. In particular, considering *specific verification scenarios*[5], in which only the participants verify the pre-signatures, our ECDSA-AS can reduce the number of zero-knowledge proofs in pre-signing phase to *one*.

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[6]. We briefly introduce our ECDSA-AS as follows: Let $(Q = xG, x)$ denote ECDSA verification key and signing key. The signer computes a pre-signing public parameter $Z = xY$ and uses x as the witness to compute $\pi_Z \leftarrow \mathsf{P}_Z((G, Q, Y, Z), x)$[7], then chooses a random value $k \leftarrow \mathbb{Z}_q$, computes $r = f(kY)$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$, and outputs the pre-signature $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$. The verification algorithm verifies π_Z and $r \stackrel{?}{=} f(\hat{s}^{-1} \cdot h(m) \cdot Y + \hat{s}^{-1} \cdot r \cdot Z)$. The adaptor algorithm takes the witness y and the pre-signature $\hat{\sigma}$ as inputs to compute $s = \hat{s} \cdot y^{-1} \bmod q$, and outputs ECDSA signature $\sigma = (r, s)$. The extraction algorithm can extract the witness by computing $y = \hat{s}/s$.

Following [2], we use "self-proving structure" $(I_Y = (Y, \pi_Y), y)$ to give security proofs. Intuitively speaking, since the zero-knowledge proof system holds straight-line extractability, the simulator can extract the witness y from the instance (Y, π_Y), then it can use ECDSA signing oracle to obtain ECDSA signature $\sigma = (r, s)$ and simulates the pre-signing oracle by computing the pre-signature $\hat{s} = s \cdot y \bmod q$.

Then, we develop two efficient ECDSA-AS schemes called ECDSA-AS$_{sk}$ and ECDSA-AS$_{wit}$ by computing the pre-signing public parameter and corresponding zero-knowledge proof offline, where ECDSA-AS$_{sk}$ uses signing key x as a witness to compute $(Z = xY, \pi_Z)$, and ECDSA-AS$_{wit}$ uses the witness y of hard relation (I_Y, y) as a witness to compute $(Z = yQ, \pi_Z)$.

Offline/online pre-signing. In ECDSA-AS [2,18], the signer computes the pre-signing public parameter $K = kY$ and proves the hard relation $((G, \hat{K} = kG, Y, K), k)$ satisfies equality of discrete logarithms $\pi_K \leftarrow \mathsf{P}_K((G, \hat{K}, Y, K), k)$, that is, there exists a witness k that is the *random value* used in the pre-signing algorithm, such that $\hat{K} = kG$ and $K = kY$.

[5] Common verification scenarios require that everyone can verify signatures. However, the pre-signature of the adaptor signature is not published on the blockchain, so it is always used in the specific verification scenarios where only the participants verify the pre-signatures off-chain and others (such as miners) need not verify pre-signatures.

[6] The zero-knowledge proof system requires straight-line extractor, also namely online extractor [12]. The straight-line extractability property allows for the extraction of a witness y for a statement Y from a proof π_Y in the random oracle model and is useful for models where the rewinding proof technique is not allowed, such as UC. [2]

[7] This zero-knowledge proof system does not require straight-line extractor. Such a proof can be derived by applying the Fiat-Shamir heuristic [11] to Chaum-Pedersen $\sum$-protocol [6] for the language comprising valid DDH tuples.

In our ECDSA-AS$_{sk}$, the signer computes $Z = xY$ 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 a witness x that is the *signing key*, such that $Q = xG$ and $Z = xY$. In our ECDSA-AS$_{wit}$, the signer[8] computes $Z = yQ$ 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 a witness y that is the *witness of hard relation* (I_Y, y), such that $Y = yG$ and $Z = yQ$. By using y as the witness, the signer (hard relation chooser) holding y can generate all pre-signing public parameters $Z_i = yQ_i$ and zero-knowledge proofs π_{Z_i} *in a batch* and *offline* for all other participants. Other participants can compute $Z_i = x_iY$ by using the signing key x_i and the instance Y.

Performance. We show the theoretical and experimental analysis of ECDSA-AS [2,18] and our ECDSA-AS$_{sk/wit}$. In the offline phase, all parties in ECDSA-AS$_{sk/wit}$ generates and checks the hard relation $I_Y = (Y, \pi_Y \leftarrow \mathsf{P}_Y(Y, y), y)$. In the online phase, the signer uses Y and $Z_i = yQ_i$ to run the online pre-signing algorithm to generate the pre-signature which can be verified by Y and $Z_i = x_iY$. Thus, the online pre-signing algorithm is similar to the original ECDSA signing algorithm except for modifying parameters by using (Z, Y) as the verification key and base point instead of (Q, G). To be specific, ECDSA-AS$_{sk/wit}$ only computes once point multiplication operation online, while ECDSA-AS in [2,18] need four times point multiplication operation. The experimental results show that ECDSA-AS$_{wit}$ reduces online pre-signing time by about 60% compared with the state-of-the-art ECDSA-AS [2] in a two-party case.

Applications. AS can be divided into off-chain and on-chain two phases. In the off-chain phase, all participants generate and verify the pre-signatures from each other, and adapt the pre-signatures into full signatures. In the on-chain phase, all participants use the time-lock to lock their coins and then publish full signatures to achieve the exchange within the timeouts. Therefore, the pre-signatures are not published on the blockchain and are only verified by the participants who satisfy special verification scenarios. Our ECDSA-AS$_{sk/wit}$ can reduce all zero-knowledge proof in the pre-signing phase, except one zero-knowledge proof of the hard relation chooser. Since other participants can compute the pre-signing public parameters $Z_i = x_iY$ and the hard relation chooser can compute the pre-signing public parameters $Z_i = yQ_i$, there is no need to use zero-knowledge proofs to ensure the correctness of pre-signing public parameters.

To our knowledge, atomic swaps are mostly for two-party exchange scenarios to ensure fairness. We consider the special batched case in which one party with many addresses (accounts) or one party with a lot of transactions that need to be exchanged with many users at once, such as the scenario of the Exchange. For this scenario, we develop *batched atomic swaps*, in which all parties first

[8] The signer can be seen as a hard relation chooser who is the protocol initiator and holds the witness y.

set the time-lock for the exchange coins on-chain[9] and then one user U_0 can exchange its coins with many users (addresses) U_i, $i \in [n]$ in a batch. Compared with running independently n times atomic swaps between U_0 and U_i, $i \in [n]$, batched atomic swaps can reduce the number of hard relations (Y, y) from n to *one*. In particular, constructing batched atomic swaps based on ECDSA-AS$_{sk/wit}$ only transmits one zero-knowledge proof, while using ECDSA-AS [2,18] requires $2n$ zero-knowledge proofs, where n denotes the number of parties in batched atomic swaps.

2 Preliminaries

2.1 Notations

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \cdots, n\}$, 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.

2.2 Hard Relation and Zero-Knowledge Proof

We recall the definition of a hard relation R with statement/witness pairs ($stat = (G, Y = yG), y$) [2]. Let L_R be the associated language defined as $\mathsf{L}_\mathsf{R} = \{(G, Y) | \exists\, y$ s.t. $((G, Y), y) \in \mathsf{R}\}$. We say that R is a hard relation if the following holds: (i) There exists a PPT sampling algorithm $\mathsf{GenR}(1^\lambda)$ that on input 1^λ outputs a statement/witness pair $((G, Y), y) \in \mathsf{R}$; (ii) The relation is poly-time decidable; (iii) For all PPT $\mathcal{A}$, the probability of $\mathcal{A}$ on input (G, Y) outputting y is negligible.

We recall the definition of a non-interactive zero-knowledge proof of knowledge (NIZKPoK) with straight-line extractors as introduced in [12]. More formally, a pair (P, V) of PPT algorithms is called a NIZKPoK with a straight-line extractor for a relation R, random oracle $\mathcal{H}$ and security parameter λ if the following holds: (i) Completeness: For any $((G, Y), y) \in \mathsf{R}$, it holds that $\mathsf{V}((G, Y), \pi \leftarrow \mathsf{P}((G, Y), y)) = 1$; (ii) Zero-knowledge: There exists a PPT simulator $\mathcal{S}$, which on input (G, Y) can simulate the proof π for any $((G, Y), y) \in \mathsf{R}$. (iii) Straight-line extractability: There exists a PPT straight-line extractor K with access to the sequence of queries to the random oracle and its answers, such that given $((G, Y), \pi)$, the algorithm K can extract the witness y with $((G, Y), y) \in \mathsf{R}$. For convenience, we omit the parameter G in this paper.

[9] All parties use time-lock to lock the exchange coins c_0 with the timeouts t_0 and c_i with the timeouts t_i, and the timeouts $t_i < t_0$, $i \in [n]$ to ensure that U_i can have enough time to react.

2.3 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:

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

Definition 1 (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:*

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

We review the existential unforgeability under chosen message attack for AS (aEUF-CMA), pre-signature adaptability, and witness extractability [2].

Definition 2 (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:*

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

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

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

Definition 3 (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* $\mathsf{pVrfy}_{vk}(m, Y, \hat{\sigma}) \rightarrow 1$*, we have* $\mathsf{Vrfy}_{vk}(m, \mathsf{Adapt}(\hat{\sigma}, y)) \rightarrow 1$.

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 [2].

Definition 4 (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:*

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

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

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

The witness extractability 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. There is one crucial difference between aWitExt and aSigForge: The adversary is allowed to choose the challenge instance Y. Hence, he knows a witness for Y and can generate a valid signature on the forgery message m. However, this is not sufficient to win the experiment aWitExt. The adversary wins only if the valid signature does not reveal a witness for Y [2].

2.4 ECDSA

We review the ECDSA scheme [1] $\sum_{\mathrm{ECDSA}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ 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 $x \leftarrow \mathbb{Z}_q$, 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 $r = f(kG)$[10] and $s = k^{-1}(h(m) + rx)$, where h is a hash function and f is defined as the projection to the x-coordinate.

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

– $\mathsf{Vrfy}_{vk}(m, \sigma) \rightarrow 0/1$. The verification algorithm computes $r' = f(s^{-1} \cdot (h(m) \cdot G + r \cdot Q))$. If $r = r' \bmod q$, outputs 1, otherwise, outputs 0.

We use the *positive* ECDSA [14,16,2] which guarantees that if (r, s) is a valid signature, then $|s| \leq (q-1)/2$, to prove the security of our ECDSA-AS.

3 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 $\sum = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. Let $(Q = xG, x)$ be the verification key and signing key of ECDSA. We define hard relations $\mathsf{R} = \{(I_Y = (Y, \pi_Y \leftarrow \mathsf{P}_Y(Y, y)), y) | \ Y = yG \wedge \mathsf{V}_Y(I_Y) = 1\}$ and $\mathsf{R}_Z = \{(I_Z = (G, Q, Y, Z), x) | Q = xG \wedge Z = xY\}$ where P_Y and V_Y denotes the proving and verification algorithm of a NIZKPoK with straight-line extractability [12], P_Z and V_Z denotes the proving and verification algorithm of a NIZK.

– $\mathsf{pSign}_{(vk,sk)}(m, I_Y) \rightarrow \hat{\sigma}$: On input a key-pair $(vk, sk) = (Q, x)$, a message m and an instance $I_Y = (Y, \pi_Y)$, the algorithm computes the pre-signing public parameter $Z = xY$, runs $\pi_Z \leftarrow \mathsf{P}_Z(I_Z = (G, Q, Y, Z), x)$, and chooses $k \leftarrow \mathbb{Z}_q$, computes $r = f(kY)$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$ and outputs the pre-signature $\hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$.
– $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \rightarrow 0/1$: On input the verification key $vk = Q$, a message m, an instance I_Y, and a pre-signature value $\hat{\sigma}$, the algorithm outputs 0, if $\mathsf{V}_Z(I_Z) \rightarrow 0$, otherwise, it computes $r' = f(\hat{s}^{-1} \cdot (h(m) \cdot Y + r \cdot Z)) \bmod q$, 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 $\sigma = (r, s)$.
– $\mathsf{Ext}(\sigma, \hat{\sigma}, I_Y) \rightarrow y$: On input the signature σ, the pre-signature $\hat{\sigma}$ and the instance I_Y, it computes $y = \hat{s}/s \bmod q$. If $(I_Y, y) \in \mathsf{R}$, it outputs y, else outputs $\perp$.

Note that in the pre-signing phase, our ECDSA-AS uses the signing key x as the witness to compute the pre-signing public parameter $Z = xY$ and zero-knowledge proof π_Z, then the later pre-signing operation is similar to original ECDSA signing algorithm except for modifying some parameters by using (Z, Y) as the verification key and base point instead of (Q, G).

Theorem 1. *Assuming that the positive ECDSA $\sum$ is SUF-CMA secure, and R is a hard relation, NIZKPoK and NIZK are secure, above ECDSA-AS $\Pi_{\mathsf{R},\sum}$ is secure in random oracle model.*

Following [2], we use self-proving structure in our ECDSA-AS and prove that our ECDSA-AS scheme satisfies pre-signature adaptability, pre-signature correctness, aEUF-CMA security, and witness extractability.

Lemma 1. (Pre-signature adaptability) *Above ECDSA-AS $\Pi_{\mathsf{R},\sum}$ 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 $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \to 1$. That is, $Y = yG, Z = xyG$, $\hat{K} = (h(m) \cdot \hat{s}^{-1})Y + r \cdot \hat{s}^{-1}Z = kY$, $r' = f(\hat{K}) = f(kY) = r$. By definition of Adapt, we know that $\mathsf{Adapt}(\hat{\sigma}, y) \to \sigma$, 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, $r' = f(K') = r$. That is, $\mathsf{Vrfy}_{vk}(m,\sigma) \to 1$.

Lemma 2. (Pre-signature correctness) *Above ECDSA-AS $\Pi_{\mathsf{R},\sum}$ satisfies pre-signature correctness.*

Proof. For any $x, y \in \mathbb{Z}_q$, $Q = xG, Y = yG$ and $m \in \{0,1\}^*$. For $\mathsf{pSign}_{(vk,sk)}(m, I_Y) \to \hat{\sigma} = (r, \hat{s}, Z, \pi_Z)$, 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 $\mathsf{pVrfy}_{vk}(m, I_Y, \hat{\sigma}) \to 1$. By Lemma 1, this implies that $\mathsf{Vrfy}_{vk}(m, \sigma) \to 1$, for $\mathsf{Adapt}(\hat{\sigma}, y) \to \sigma = (r, s)$. By the definition of Adapt, we know that $s = \hat{s} \cdot y^{-1}$ and $y' = \mathsf{Ext}(\sigma, \hat{\sigma}, I_Y) = \hat{s}/s = \hat{s}/(\hat{s}/y) = y$. Hence, $(I_Y, y') \in \mathsf{R}$.

Lemma 3. (aEUF-CMA security) *Assuming that the positive ECDSA signature scheme $\sum$ is SUF-CMA secure, R is a hard relation, NIZKPoK and NIZK are secure, above ECDSA-AS $\Pi_{\mathsf{R},\sum}$ is aEUF-CMA secure.*

Proof. We prove the aEUF-CMA security by reduction to the strong unforgeability of positive ECDSA signatures. Following [2], our ECDSA-AS uses the same hard relation $(I_Y = (Y, \pi_Y), y)$, where NIZKPoK$_Y$ satisfies straight-line extractability, so the simulator can extract the witness from I_Y. Our proof works by showing that, for any PPT adversary $\mathcal{A}$ breaking aEUF-CMA security of the ECDSA-AS, we construct a PPT simulator $\mathcal{S}$ who breaks 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 $\mathcal{H}_{\text{ECDSA}}$. It needs to simulate oracle for $\mathcal{A}$, namely random oracle ($\mathcal{H}$), signing oracle ($\mathcal{O}_{\text{Sign}}$) and pre-signing oracle ($\mathcal{O}_{\text{pSign}}$).

The simulator $\mathcal{S}$ can use its oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ and $\mathcal{H}_{\text{ECDSA}}$ to simulate $\mathcal{O}_{\text{Sign}}$ and $\mathcal{H}$. The main challenge is simulating $\mathcal{O}_{\text{pSign}}$ queries. Because $\mathcal{S}$ can extract the witness from I_Y, it uses its oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ to get a full signature on m which is queried by $\mathcal{A}$, and transform the full signature into a pre-signature. What's more, $\mathcal{S}$ can use the zero-knowledge property of NIZK$_Z$ to simulate π_Z for a statement (G, Q, Y, Z) without knowing the corresponding witness x.

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.

– Game G_1: This game works as G_0 with the exception that upon the adversary outputting a forgery σ^*. It checks that if completing the pre-signature $\hat{\sigma}$ using the secret value y results in σ^*. If yes, it aborts.
– Game G_2: This game works as G_1 excepting that in $\mathcal{O}_{\text{pSign}}$, it extracts a witness y' by executor K. It aborts if $(I_Y, y') \notin \mathsf{R}$.
– Game G_3: This game works as G_2 excepting that it extracts a witness y and calculates $Z = yQ$, and simulates a zero-knowledge proof π_S.
– Game G_4: In this game, upon receiving the challenge message m^* from $\mathcal{A}$, it 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.

– Signing oracle queries: Upon $\mathcal{A}$ querying $\mathcal{O}_{\text{Sign}}$ 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 $\mathcal{H}$ 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: Upon $\mathcal{A}$ querying $\mathcal{O}_{\text{pSign}}$ on input (m, I_Y), the simulator extracts y, and forwards m to $\mathcal{O}_{\text{ECDSA-sign}}$ and gets (r, s), then $\mathcal{S}$ computes $\hat{s} = s \cdot y$, $Z = yQ = xY$ and simulates a zero-knowledge proof π_S, and outputs $(r, \hat{s}, Z, \pi_S)$.
– In the challenge phase: Upon $\mathcal{A}$ outputting the challenge message m^*, $\mathcal{S}$ generates $(I_Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$, forwards m^* to $\mathcal{O}_{\text{ECDSA-sign}}$ and gets (r, s). And then, $\mathcal{S}$ generates the pre-signature $\hat{\sigma}^*$ in the same way as during $\mathcal{O}_{\text{pSign}}$. Upon $\mathcal{A}$ outputting σ^*, the simulator outputs (m^*, σ^*) as its own forgery.

Therefore, the simulator $\mathcal{S}$ can simulate the views of $\mathcal{A}$. It remains to show that the forgery output by $\mathcal{A}$ can be used by the simulator to win the 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. The simulator gets a challenge I_Y^*, and it generates a key pair $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ to simulate $\mathcal{A}$s queries of $\mathcal{H}$, $\mathcal{O}_{\text{Sign}}$ and $\mathcal{O}_{\text{pSign}}$. This simulation of the oracles works as described in G_1. 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 $\hat{\sigma}$ to $\mathcal{A}$ who outputs a forgery σ^*.

Assuming that Bad_1 happened (i.e. $\mathsf{Adapt}(\hat{\sigma}, y) = \sigma^*$), the simulator can extract $y^* \leftarrow \mathsf{Ext}(\sigma^*, \hat{\sigma}, I_Y^*)$. 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.

Claim 2 *G_0, G_1, G_2, G_3 and G_4 are computationally indistinguishable.*

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

According to the straight-line extractability of the NIZKPoK$_Y$, for a witness y extracted from a proof π_Y of the instance I_Y such that $\mathsf{V}_Y(I_Y, \pi_Y) \to 1$, it holds that $(I_Y, y) \in \mathsf{R}$ except with negligible probability. It holds that $|\Pr[G_2 = 1] - \Pr[G_1 = 1]| \leq \mathsf{negl}_2(\lambda)$.

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

Following the above proof, due to the zero-knowledge property of the NIZK$_Z$, G_4 is indistinguishable from G_3 and it holds that $|\Pr[G_4 = 1] - \Pr[G_3 = 1]| \leq \mathsf{negl}_2(\lambda)$.

Claim 3 (m^*, σ^*) *constitutes a valid forgery in positive ECDSA strongSigForge game.*

Proof. We show that (m^*, σ^*) has not been output by the oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ before. Note that $\mathcal{A}$ 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:

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

Lemma 4. (Witness extractability). *Assuming that the positive ECDSA is SUF-CMA secure,* R *is a hard relation, NIZKPoK and NIZK are secure, above ECDSA-AS* $\Pi_{\mathsf{R},\sum}$ *is witness extractable.*

Proof. Our proof is to reduce the witness extractability to the strong unforgeability of the positive ECDSA. Following the proof of Lemma 3, the simulator $\mathcal{S}$ can use its oracle $\mathcal{O}_{\text{ECDSA-Sign}}$ and $\mathcal{H}_{\text{ECDSA}}$ to simulate $\mathcal{O}_{\text{Sign}}$ and $\mathcal{H}$ of $\mathcal{A}$.

The main challenge in this proof is to simulate the pre-signing oracle $\mathcal{O}_{\text{pSign}}$. The crucial difference between aWitExt and aSigForge is that in the challenge phase of aSigForge, I_Y is chosen by a challenger, but in the challenge phase of aWitExt, I_Y is chosen by $\mathcal{A}$. That is, $\mathcal{S}$ can not choose (I_Y, y). Following [2], our ECDSA-AS uses the same hard relation $(I_Y = (Y, \pi_Y), y)$, where NIZKPoK$_Y$ satisfies straight-line extractability, so $\mathcal{S}$ can extract the witness y from challenge instance $I_Y = (Y, \pi_Y)$. And then, $\mathcal{S}$ forwards m to $\mathcal{O}_{\text{ECDSA-sign}}$ and gets the

signature $\sigma = (r, s)$, then $\mathcal{S}$ computes $\hat{s} = s \cdot y$, $Z = yQ$ and simulates a zero-knowledge proof π_S, and outputs the pre-signature $\hat{\sigma} = (r, \hat{s}, Z, \pi_S)$.

Therefore, we can construct a simulator $\mathcal{S}$ following the proof of Lemma 3 excepting that in the challenge phase, $\mathcal{S}$ does not generate the hard relation (I_Y, y) to get the witness y, but obtains the witness from the instance I_Y chosen by $\mathcal{A}$ based on the straight-line extractability. $\mathcal{S}$ can simulate the views of $\mathcal{A}$. The simulator can win the positive ECDSA strongSigForge game if $\mathcal{A}$ can break the witness extractability of ECDSA-AS.

4 Fast ECDSA-based Adaptor Signature Schemes with Offline/Online Pre-signing

In this section, we show two fast ECDSA-AS schemes called ECDSA-AS$_{sk}$ and ECDSA-AS$_{wit}$ with offline/online pre-signing, where ECDSA-AS$_{sk}$ uses the signing key x as the witness to compute $Z = xY$, and ECDSA-AS$_{wit}$ uses the witness y of hard relation (I_Y, y) as the witness to compute $Z = yQ$.

In our ECDSA-AS, the pre-signing public parameter $Z = xY$ and the zero-knowledge proof $\pi_Z \leftarrow \mathsf{P}_Z(I_Z = (G, Q, Y, Z), x)$ are independent of the message m and the random value k, so the signer can compute the pre-signing public parameter and the zero-knowledge proof *offline* before getting the message. ECDSA-AS$_{sk}$ can be designed from our ECDSA-AS directly with offline computing $Z = xY$ and π_Z. Refer to the section 3 for specific construction which is ignored here.

We construct efficient ECDSA-AS$_{wit}$ as follows. Formally, Let $(Q = xG, x)$ be the verification key and signing key of ECDSA. We define hard relations $\mathsf{R} = \{(I_Y = (Y, \pi_Y \leftarrow \mathsf{P}_Y(Y, y)), y) |\ Y = yG \wedge \mathsf{V}_Y(I_Y) = 1\}$, $\mathsf{R}_Z = \{(I_Z = (G, Y, Q, Z), y) | Y = yG \wedge Z = yQ\}$ and $I = (I_Y, I_Z)$.

- $\mathsf{pSign}_{(vk,sk)}(m, I) \rightarrow \hat{\sigma}$: On input a key-pair $(vk, sk) = (Q, x)$, a message m and an instance I, the algorithm chooses $k \leftarrow \mathbb{Z}_q$, computes $r = f(kY)$, $\hat{s} = k^{-1}(h(m) + rx) \bmod q$ and outputs $\hat{\sigma} = (r, \hat{s})$.
- $\mathsf{pVrfy}_{vk}(m, I, \hat{\sigma}) \rightarrow 0/1$: On input the verification key $vk = Q$, a message m, an instance I, and a pre-signature value $\hat{\sigma}$, the algorithm 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 $\sigma = (r, s)$.
- $\mathsf{Ext}(\sigma, \hat{\sigma}, I) \rightarrow y$: On input the signature σ, the pre-signature $\hat{\sigma}$ and the instance I, it computes $y = \hat{s}/s \bmod q$. If $(I, y) \in \mathsf{R}$, it outputs y, else outputs $\perp$.

Note that ECDSA-AS$_{wit}$ is similar to our ECDSA-AS excepting that the signer can compute $Z = yQ$ and $\pi_Z \leftarrow \mathsf{P}_Z(I_Z = (G, Y, Q, Z), y)$ offline. Before running the online pre-signing algorithm, the signer should *check the validity* of π_Y and π_Z offline to ensure that Y and Z are correct.

Correctness. Following the proofs of lemma 1 and lemma 2, our ECDSA-AS$_{sk/wit}$ schemes also satisfy pre-signature adaptability and pre-signature correctness.

Security. Our ECDSA-AS$_{sk/wit}$ schemes embed the hard relation $(I_Y = (Y, \pi_Y), y)$ [2]. In the security proof, the simulator can extract the witness y and simulate the pre-signing oracle. Following the proofs of lemma 3 and lemma 4, our ECDSA-AS$_{sk/wit}$ schemes also satisfy aEUF-CMA security and witness extractability.

Comparisons with ECDSA-AS [2]. Our ECDSA-AS$_{wit}$ use y as the witness, so the signer (hard relation chooser) can help all other participants compute the pre-signing public parameter $Z_i = yQ_i$ and the zero-knowledge proofs π_{Z_i} in a batch and offline. In particular, consider the special verification scenario, such as (batched) atomic swaps, ECDSA-AS$_{wit}$ only transmits *one* zero-knowledge proof π_{Z_0} which is independent of the number of participants, since all participants can compute the pre-signing public parameter locally.

Our ECDSA-AS$_{sk/wit}$ can be seen as an adaptor signature that specifies the signer, since the hard relation chooser requires the verification key Q_i of other parties to compute $Z_i = yQ_i$ and π_{Z_i}, while ECDSA-AS [2] does not restrict the signer's verification key. However, this does not affect the application of ECDSA-AS$_{sk/wit}$ in atomic swaps, because the verification keys are public before the protocol begins. In addition, consider special verification scenarios, ECDSA-AS$_{sk/wit}$ can remove the above restriction because the zero-knowledge proofs of other parties can be removed.

5 Performance and Experimental Results

5.1 Theoretical Analysis

As is shown in Table 1, we give the theoretical analysis of communication cost and efficiency of ECDSA-AS [2,18] and our schemes, respectively. The first ECDSA-AS proposed by Moreno-Sanchez et al. [18] does not provide provable security. Then Aumayr et al. [2] uses self-proving structure $(I_Y = (Y, \pi_Y), y)$ to give a provably secure ECDSA-AS. But this scheme requires that proving $\hat{K} = kG$ and $K = kY$ satisfy equality of discrete logarithms with the witness k. For each message to be signed, the signer needs to choose new random values and computes new pre-signing public parameters and zero-knowledge proofs.

Our ECDSA-AS$_{sk/wit}$ can use the witness x or y to prove $(Z = xY, Q = xG)$ or $(Z = yQ, Y = yG)$ satisfy equality of discrete logarithms offline. The online pre-signing algorithm is similar to ECDSA signing algorithm and can enjoy the same efficiency as ECDSA. In ECDSA-AS$_{wit}$, the hard relation chooser can compute all pre-signing public parameters $Z_i = yQ_i$ and zero-knowledge proofs for all other participants *in a batch* and *offline*. In particular, consider special verification scenarios, such as (batched) atomic swaps, ECDSA-AS$_{sk/wit}$ can reduce the number of zero-knowledge proofs to one, since all parties can compute public pre-signing parameters locally, but ECDSA-AS [2,18] requires the number of zero-knowledge proofs is linear with the number of participants.

Table 1: Communication cost and efficiency comparison

Schemes	PK size	SK size	Online pre-signature size	The number of zk proofs	The number of pre-signing parameter	Batched pre-signing parameter	Provable security
ECDSA-AS [18]	$\|\mathbb{G}\|$	$\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|+4\|\mathbb{Z}_q\|$	$2n$	$2n$	×	?
ECDSA-AS [2]	$\|\mathbb{G}\|$	$\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|+4\|\mathbb{Z}_q\|$	$2n$	$2n$	×	√
Our ECDSA-AS$_{sk}$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_q\|$	$2\|\mathbb{Z}_q\|$	n	$n+1$	×	√
Our ECDSA-AS$_{wit}$	$\|\mathbb{G}\|$	$\|\mathbb{Z}_q\|$	$2\|\mathbb{Z}_q\|$	1	$n+1$	√	√

‡ $|\mathbb{G}|$ and $|\mathbb{Z}_q|$ denotes the size of the element in the group $\mathbb{G}$ and $\mathbb{Z}_q$, respectively. n denotes the number of parties in the one-to-n atomic swaps. ? denotes unclear.

5.2 Experimental Analysis

In order to evaluate the practical performance of our schemes, we implement the ECDSA-AS [2], our ECDSA-AS, and ECDSA-AS$_{wit}$ based on the OpenSSL library. All experiments are carried out on an Intel Core i5 CPU 2.3 GHz and 8GB RAM running macOS High Sierra 10.13.3 system.

We run ECDSA-AS [2], our ECDSA-AS and ECDSA-AS$_{wit}$ on the standard NIST curve NID.X9.62.prime256v1. Since the verification algorithm, the adaptor algorithm, and the extraction algorithm are roughly same, we omit the comparison. We show the efficiency of the online pre-signing algorithm in Table 2. The average running times over 1000 executions of the online pre-signing operation in ECDSA-AS [2], our ECDSA-AS and ECDSA-AS$_{wit}$ are 173.65 μs, 189.72 μs and 71.64 μs. The experimental results show that our ECDSA-AS$_{wit}$ reduces online pre-signing time by about 60% compared with the state-of-the-art ECDSA-AS [2].

Table 2: Runtime of the online pre-signing operation comparing our ECDSA-AS and ECDSA-AS$_{wit}$ to ECDSA-AS [2]

Schemes	ECDSA-AS [2]	our ECDSA-AS	our ECDSA-AS$_{wit}$
Runtime (online)	173.65 μs	189.72 μs	71.64 μs

6 Application

6.1 Verification Scenario

According to the definition of AS [2], the verification of pre-signature does not limit the verifier, so it can be verified by *anyone*. However, the pre-signature of AS is generated and verified off-chain and is not published on the blockchain, so AS does not require such a strong property and the pre-signature satisfies the specific verification scenario in which it is only verified by the participants.

As mentioned in Figure 1, in the atomic swaps, the pre-signatures are only generated and verified by participants U_0 and U_1 off-chain. Thus, ECDSA-AS$_{sk/wit}$ can reduce the number of zero-knowledge proofs. To be specific, the zero-knowledge proof π_{Z_1} of U_1 can be removed, because U_0 can use the witness y to compute $Z_1 = yQ_1$. The zero-knowledge proof π_{Z_0} of U_0 cannot be removed, since U_1 does not know the signing key x_0 and the witness y of U_0, and requires π_{Z_0} to ensure U_0 generates Z_0 correctly. In particular, even if U_0 runs an atomic swap protocol with many parties U_i, $i \in [n]$, it still only needs one zero-knowledge proof π_{Z_0}. Note that these modifications do not affect correctness or security, since the simulator can extract the witness y from I_Y and simulates the pre-signing public parameter $Z_1 = yQ_1$.

6.2 Batched Atomic Swaps

We develop batched atomic swaps from one-to-one atomic swaps: U_0 (hard relation chooser) spends c_{0i} (transaction tx_{0i}) to U_i, $i \in [n]$ in a batch and U_i spends each c_i (transaction tx_i) to U_0. It can be applied to one party with many addresses (accounts) or one party with a lot of transactions that need to be exchanged with many users once, such as the scenario of the Exchange. Compared with running independently n times one-to-one atomic swaps between U_0 and U_i, $i \in [n]$, batched atomic swaps reduce the number of hard relations (Y, y) from n to *one*.

We introduce batched atomic swaps as follows: All parties U_0 and U_i, $i \in [n]$ first set the time-lock for c_{0i} and c_i on-chain, where the timeouts $t_i < t_0$ such that U_i can have enough time to react. Then, U_0 chooses one hard relation $(Y, y) \in R$ and pre-signing the transactions tx_{0i} for spending the coins c_{0i} to U_i in a batch, and sends the pre-signature $\hat{\sigma}_{0i}$, tx_{0i}, Y to U_{0i}. Then, U_i checks the validity of $\hat{\sigma}_{0i}$ and pre-signing a transaction tx_i for spending the coins c_i to U_0 and sends the pre-signature $\hat{\sigma}_i$, tx_i to U_0. U_0 can check the validity of all pre-signatures $\hat{\sigma}_i$[11] and adapts $\hat{\sigma}_i$ into the full signature σ_i by the witness y, and publishes all σ_i on the blockchain to get the coin c_i in a batch. U_i can extract the witness y from σ_i and $\hat{\sigma}_i$ and adapts $\hat{\sigma}_{0i}$ into σ_{0i}, and publishes σ_{0i} on the blockchain to get the coin c_{0i} before timeouts t_0.

Constructions based on ECDSA-AS. Our ECDSA-AS$_{wit}$ is more efficient for using in batched atomic swaps than ECDSA-AS [2,18]. We show the batched atomic swap protocol based on ECDSA-AS$_{wit}$ in Figure 2, and give the comparison of n times independent ECDSA-based atomic swaps and once ECDSA-based batched atomic swap in Figure 3. To be specific, in the offline phase, U_0 computes $Z_0 = yQ_0$, $\pi_{Z_0} \leftarrow \mathsf{P}_Z((G, Y, Q_0, Z_0), y)$, and n pre-signing public parameters $Z_i = yQ_i$ in a batch and offline. U_i checks $\mathsf{V}_Z((G, Y, Q_0, Z_0), \pi_{Z_0}) \to 1$ and computes $Z_i = x_iY$. In the online phase, U_0 runs n times AS$_{wit}$.pSign and AS$_{wit}$.pVrfy and each U_i runs once AS$_{wit}$.pSign and AS$_{wit}$.pVrfy, where

[11] U_0 must check all pre-signatures, because any full signature is published on blockchain, the witness y can be extracted, and all coins can be taken.

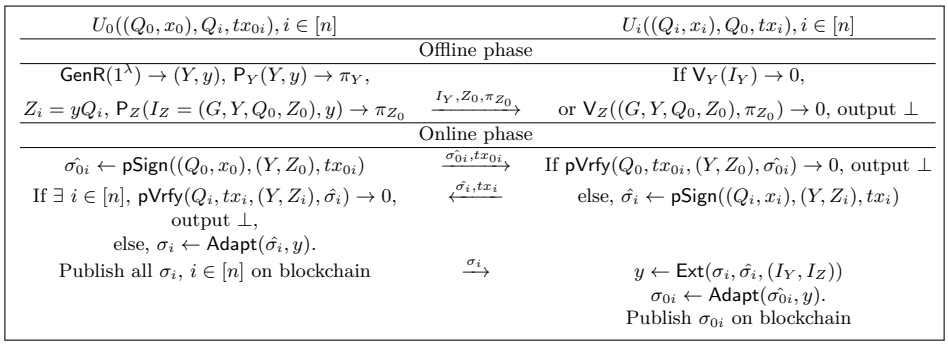

Fig. 2: Batched atomic swap based on ECDSA-AS$_{wit}$

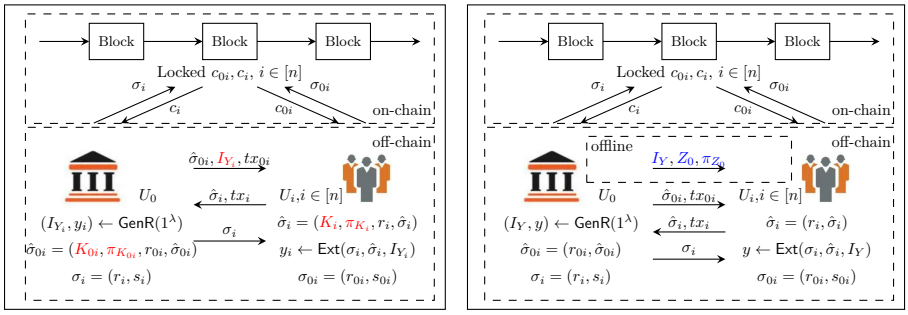

Fig. 3: n times independent ECDSA-based atomic swaps (left) and once ECDSA-based batched atomic swap (right)

Table 3: Comparison of ECDSA-AS [2,18] and our ECDSA-AS$_{sk/wit}$ used in (batched) atomic swaps

Schemes	Users	Atomic swaps ($i = 1$)			Batched atomic swaps ($i \in [n]$)		
		Hard relation	Pre-signing		Hard relations	Pre-signing	
			Offline	Online		Offline	Online
ECDSA-AS [18]	U_0	(Y_i, y_i)	—	$\hat{\sigma_0} = (K_0, \pi_{K_0}, r_0, \hat{s_0})$	(Y, y)	—	$\hat{\sigma_{0i}} = (K_{0i}, \pi_{K_{0i}}, r_{0i}, \hat{s_{0i}})$
	U_i	—	—	$\hat{\sigma_i} = (K_i, \pi_{K_i}, r_i, \hat{s_i})$	—	—	$\hat{\sigma_i} = (K_i, \pi_{K_i}, r_i, \hat{s_i})$
ECDSA-AS [2]	U_0	(I_{Y_i}, y_i)	—	$\hat{\sigma_0} = (K_0, \pi_{K_0}, r_0, \hat{s_0})$	(I_Y, y)	—	$\hat{\sigma_{0i}} = (K_{0i}, \pi_{K_{0i}}, r_{0i}, \hat{s_{0i}})$
	U_i	—	—	$\hat{\sigma_i} = (K_i, \pi_{K_i}, r_i, \hat{s_i})$	—	—	$\hat{\sigma_i} = (K_i, \pi_{K_i}, r_i, \hat{s_i})$
Our ECDSA-AS$_{sk}$	U_0	(I_{Y_i}, y_i)	$Z_0 = x_0 Y, \pi_{Z_0}$	$\hat{\sigma_0} = (r_0, \hat{s_0})$	(I_Y, y)	$Z_0 = x_0 Y, \pi_{Z_0}$	$\hat{\sigma_{0i}} = (r_{0i}, \hat{s_{0i}})$
	U_i	—	$Z_i = x_i Y$	$\hat{\sigma_i} = (r_i, \hat{s_i})$	—	$Z_i = x_i Y$	$\hat{\sigma_i} = (r_i, \hat{s_i})$
Our ECDSA-AS$_{wit}$	U_0	(I_{Y_i}, y_i)	$Z_0 = yQ_0, Z_i = yQ_i, \pi_{Z_0}$	$\hat{\sigma_0} = (r_0, \hat{s_0})$	(I_Y, y)	$Z_0 = yQ_0, Z_i = yQ_i, \pi_{Z_0}$	$\hat{\sigma_{0i}} = (r_{0i}, \hat{s_{0i}})$
	U_i	—	—	$\hat{\sigma_i} = (r_i, \hat{s_i})$	—	—	$\hat{\sigma_i} = (r_i, \hat{s_i})$

‡ — denotes no such operation, Q_i denotes the verification key, Z_i and K_i denote the pre-signing public parameters of our ECDSA-AS$_{sk/wit}$ and ECDSA-AS [18,2], π_{Z_i} and π_{K_i} denote the zero-knowledge proofs of proving Z_i and K_i are generated correctly. n denotes the number of parties in batched atomic swaps. $Z_i = x_i Y$ and $Z_i = yQ_i$, $i \in [n]$ don't need to be transmitted.

AS$_{wit}$.pSign and AS$_{wit}$.pVrfy is same as original ECDSA signing and verification algorithms.

In ECDSA-AS [2,18], each user uses the random value k as the witness to generate the pre-signing public parameter $K = kY$ and zero-knowledge proof π_K. For a batch of transactions, U_0 needs to choose n random values to generate individual pre-signing public parameter $K_{0i} = k_{0i}Y$ and zero-knowledge proof $\pi_{K_{0i}} \leftarrow \mathsf{P}_K(G, \hat{K_{0i}} = k_{0i}G, Y, K_{0i})$ for U_i, and U_i uses different random value k_i to compute pre-signing public parameter $K_i = k_iY$ and zero-knowledge proof $\pi_{K_i} \leftarrow \mathsf{P}_K(G, \hat{K_i} = k_iG, Y, K_i)$. What's more, U_0 needs to check the validity of all n proofs π_{K_i}, and U_i also needs to check the validity of $\pi_{K_{0i}}$.

As depicted in Table 3, we give a comparison of (batched) atomic swaps based on ECDSA-AS [2,18] or ECDSA-AS$_{sk/wit}$. Batched atomic swaps based on ECDSA-AS [2,18] need to compute $2n$ pre-signing public parameters and $2n$ zero-knowledge proofs. The batched atomic swaps can be seen as specific verification scenarios. Since U_0 computes all pre-signing public parameters $Z_i = yQ_i$ and U_i computes each pre-signing public parameter $Z_i = x_iY$ locally, batched atomic swaps based on ECDSA-AS$_{wit}$ only requires one zero-knowledge proof π_{Z_0}. Compared with [2,18], ECDSA-AS$_{wit}$ reduces $2n-1$ zero-knowledge proofs.

7 Conclusion

In this paper, we propose an ECDSA-based adaptor signature and give the security proof based on ECDSA. And then, we develop two ECDSA-AS schemes called ECDSA-AS$_{sk}$ and ECDSA-AS$_{wit}$ with offline/online pre-signing which are more efficient than the state-of-the-art ECDSA-AS [2]. In particular, considering specific verification scenarios, ECDSA-AS$_{wit}$ reduces the number of zero-knowledge proofs in the pre-signing phase to one, independent of the number of participants. Furthermore, we develop batched atomic swaps which can reduce the number of hard relations in a batch compared with independently running one-to-one atomic swaps. Finally, we use our ECDSA-AS$_{wit}$ to construct the batched atomic swaps, it can reduce the number of zero-knowledge proofs into one compared with [2,18].

Acknowledgements. We thank the anonymous reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program of China (Grant No. 2021YFA1000600) and the National Natural Science Foundation of China (Grant No. 62272269).

References

1. American National Standards Institute: X9.62: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa) (2005)
2. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: ASIACRYPT 2021. pp. 635–664. Springer (2021)

3. Aumayr, L., Maffei, M., Ersoy, O., Erwig, A., Faust, S., Riahi, S., Hostáková, K., Moreno-Sanchez, P.: Bitcoin-compatible virtual channels. In: 42nd IEEE Symposium on Security and Privacy, SP 2021. pp. 901–918 (2021)
4. Bitcoin Wiki: Payment channels (2018), https://en.bitcoin.it/wiki/Paymentchannels
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001. pp. 136–145. IEEE Computer Society (2001)
6. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92. Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer (1992)
7. 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. pp. 3–18. Springer (2015)
8. Deshpande, A., Herlihy, M.: Privacy-preserving cross-chain atomic swaps. In: Financial Cryptography and Data Security - FC 2020 International Workshops. Lecture Notes in Computer Science, vol. 12063, pp. 540–549. Springer (2020)
9. Eckey, L., Faust, S., Hostáková, K., Roos, S.: Splitting payments locally while routing interdimensionally. IACR Cryptol. ePrint Arch. **2020**, 555 (2020)
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. Lecture Notes in Computer Science, vol. 12309, pp. 378–397. Springer (2020)
11. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO '86. pp. 186–194 (1986)
12. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 152–168. Springer (2005)
13. Gugger, J.: Bitcoin-monero cross-chain atomic swap. IACR Cryptol. ePrint Arch. **2020**, 1126 (2020)
14. Lindell, Y.: Fast secure two-party ECDSA signing. In: Advances in Cryptology - CRYPTO 2017. pp. 613–644 (2017)
15. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017. pp. 455–471. ACM (2017)
16. 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
17. 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. Lecture Notes in Computer Science, vol. 11598, pp. 508–526. Springer (2019)
18. 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
19. Poelstra, A.: Lightning in scriptless scripts. mimblewimble team mailing list (2017), https://lists.launchpad.net/mimblewimble/msg00086.html
20. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf

21. Schnorr, C.: Efficient identification and signatures for smart cards. In: Advances in Cryptology - CRYPTO '89. pp. 239–252 (1989)