Technical Report on Threshold ECDSA in the Preprocessing Setup

Manuel B. Santos, Miguel de Vega, Wicher Malten, Mehmet Ugurbil, and Nathalie Regnault

Nillion

November 30, 2023

Abstract

Threshold ECDSA (Elliptic Curve Digital Signature Algorithm) has garnered significant attention from both industry and research, primarily due to its blockchain applications, especially in the context of threshold wallets. Existing solutions often assume that the client is either an integral part of the signers, as evident in self-custodial or non-custodial wallets, or that they entirely delegate the signing procedure to an external network, as observed in custodial wallets. The former allows clients to defend against collusion attacks from other signers, while the latter enables more efficient online phases through the use of preprocessing material. In this report, we explore the development of a threshold signature scheme that keeps some of the key benefits from both custodial and non-custodial settings. Specifically, we present a threshold ECDSA signature scheme in the client-server model that combines the optimal online round complexity from the custodial setting with resilience against collusion among all signer nodes during the online phase, as observed in non-custodial wallets. The scheme is also secure against user impersonation attacks, a key concern with custodial wallets.

1 Introduction

Threshold cryptography distributes information and responsibility among multiple parties [5, 12, 25], enhancing security and resilience. A prominent application of this approach is threshold signature schemes (TSS), where a set of n parties possessing shares of the signing key can sign a message M only if at least t parties agree to sign [11]. By decentralizing the signing process through a network of diverse entities, this strategy mitigates the risk of single points of failure. Recently, (TSS) have been adopted within the Web3 ecosystem [19, 9] as a measure to enhance security and mitigate inherent risks associated with client-driven key management. Precisely, TSS schemes allow wallets to operate within three distinct scenarios: custodial, wherein all signing key shares are distributed among a designated set of signing parties; self-custodial, whereby the client exclusively manages all cryptographic keys; and non-custodial, involving the distribution of signing key shares between the client and designated signing parties [9].

Custodial wallets present operational advantages over non-custodial counterparts: they delegate key management responsibilities to external service providers staffed by security experts, thereby reducing the risk of theft or loss. Additionally, from an efficiency standpoint, custodial wallets do not necessitate client involvement in the signing process, allowing preprocessing tasks to be efficiently handled by the network before signing. However, custodial wallets exhibit vulnerability to full collusion among signers, whether for signing new messages or disclosing shared secret keys. This vulnerability is addressed by non-custodial wallets, where the client actively participates in the signing process. Moreover, entrusting key management responsibilities in custodial setups introduces the risk of user impersonation attacks, wherein an attacker seizes control of the user's wallet by posing as the legitimate user. Consequently, there exists a desire to combine the benefits of both custodial and non-custodial solutions. While TSS protocols maintain transparency regarding the roles of clients and signing parties, enabling execution in both custodial and non-custodial settings, they lack the flexibility for threshold wallets to optimize advantages from both approaches. Therefore, one might wonder:

Is it possible to develop a TSS protocol that keeps some of the key benefits of both custodial and non-custodial settings?

In this report, we present a threshold ECDSA signature scheme in the client-server model [21] that keeps the operational benefits of a custodial setting while eliminating impersonation attacks and approximating its trust assumptions to the unforgeability robustness of the non-custodial setting. This is realized through the encryption of precomputed material linked to the secret key and a white-box application of a secure multiparty computation (MPC) protocol [27], recently introduced and instantiated using additive shares.

Related work. Some signature schemes, such as BLS and Schnorr's scheme, can be easily adapted to the threshold setting [4]. However, the elliptic curve digital signature algorithm (ECDSA) has received more research attention due to the challenges of executing non-linear operations over shares. Recent papers [20, 15, 14, 6, 8, 13] mainly focus on dishonest majority scenarios with statically corrupted parties, which is suitable enough for practical deployment as highlighted in [1]. Nonetheless, these approaches often rely on additional assumptions, such as strong RSA [14, 6] and utilize complex building blocks such as Multiplicative-to-Additive (MtA) shares [14, 6, 13]. These factors increase the complexity of deployments, thereby making them vulnerable to potential attacks [2, 26, 24].

Optimization efforts primarily target reducing communication rounds in TSS protocols, which heavily rely on network latency [1]. The most efficient protocols, such as [6, 8], operate in a preprocessing setup, where a presignature is generated, enabling a non-interactive online phase. However, these protocols do not address a scenario where the client does not own a share of the signing key while having the ability to prevent signers from signing independently under full collusion.

Our contribution. We introduce a threshold ECDSA scheme in the dishonest majority setting with abort within the client-server model, creating sufficient preprocessing material to facilitate an optimal online phase — comprising one round for the client to transmit a message to the servers and another round to receive the result. Our protocol is unforgeable with $n-1$ corruptions and with full collusion among signers during the client dependent phase of the distributed signing protocol. Furthermore, it is secure against user impersonation attacks.

High-level picture of the scheme. The proposed scheme is tailored for deployment in a network architecture where individual clients are connected with a group of n signers. In essence, the network holds encrypted shares related to the signing secret key, while the clients possess the secret key of the corresponding encryption scheme. This configuration empowers the client to prevent unauthorized attempts by the signers to forge signatures.

We follow the MPC protocol outlined in [27]. This protocol calculates multivariate polynomials without interaction during the computation phase. Specifically, the protocol operates on elements of the form $\langle x \rangle_\lambda := x \cdot h^{-\lambda}$ where λ is secret shared between the parties. Within the context of this report, we refer to $\langle x \rangle_\lambda$ as a masked value of x with respect to the masking exponent λ using public generator h. It is worth noting that this construction essentially constitutes a multiplicative one-time pad in the multiplicative group $\mathbb{Z}_q^*$, for prime q.

The signature scheme comprises two protocols: a distributed key generation (DKG) protocol, denoted as $\pi_{\mathtt{DKG}}$, and a threshold signing protocol, denoted as $\pi_{\mathtt{Sign}}$. To ensure compatibility with the underlying MPC protocol [27], the DKG protocol outputs an encrypted signing secret key share tuple $(\langle x \rangle_\lambda, \mathbf{e}_{\mathsf{pk},\lambda})$ alongside the conventional signing public key $y = x \cdot G$, where G represents an elliptic curve generator. Here, $\mathbf{e}_{\mathsf{pk},\lambda} := \mathsf{Enc}_{\mathsf{pk}}([\![\lambda]\!])$, with $[\![\cdot]\!]$ denoting the additive secret sharing scheme in $\mathbb{Z}_{q-1}$ and $\mathsf{Enc}_{\mathsf{pk}}$ representing an additively homomorphic scheme under the public key pk and secret key sk owned by the client. The generation of the random masked value $\langle x \rangle_\lambda$ and the corresponding public key y is accomplished using standard MPC primitives combined with preprocessing material from the MPC protocol [27]. This involves the reconstruction of additive shares both in the exponent of $h \in \mathbb{Z}_q^*$ and in the coefficient of a chosen elliptic curve generator G.

The threshold signing protocol, denoted as $\pi_{\mathtt{Sign}}$, is designed to compute the ECDSA signature equation. This equation takes the form of a polynomial ($s = k^{-1} \cdot m + k^{-1} \cdot x \cdot r$), where all elements are in the field $\mathbb{Z}_q$. The signing protocol is structured into three distinct phases:

1. Signers preprocessing phase: This phase involves computing preprocessing material without requiring any client intervention. An interesting optimization in this phase is the local computation of inversions, utilizing the property $\langle x \rangle_\lambda^{-1} = \langle x^{-1} \rangle_{-\lambda}$. This phase assumes at least one signer is honest.

2. Client preprocessing phase: This phase consists of a single round where signers reveal two masking exponents to the client. This phase tolerates full signer corruption.

3. Online phase: This phase encompasses two rounds. In the first round, the client sends the message to be signed, and in the second round, the client receives the result from the signers. It is noteworthy that during the online phase, no communication occurs among the signers, resembling the evaluation phase of the MPC protocol as detailed in [27, Section 2.2]. This phase also tolerates full signer corruption.

2 Technical overview

2.1 ECDSA Signing

We recall that ECDSA is parametrized by an elliptic curve over a finite field and defined by a triple $(\mathbb{G}, G, q)$, where $\mathbb{G}$ is an additive subgroup of points in the curve with prime order q and generator G. Given message M, secret key x and random nonce k in $\mathbb{Z}_q^*$, the ECDSA signature is a pair $(s, r) \in \mathbb{Z}_q^*$ where $s = k^{-1} \cdot (H(M) + x \cdot r)$, for a cryptographic hash function H. Here, r is the x-coordinate of the point $k \cdot G$, i.e. $(r, \_) := k \cdot G$.

2.2 Secret sharing

A $(t, n)-$threshold secret sharing scheme is a pair of algorithms $(\mathsf{Share}, \mathsf{Reconstruct})$. The Share algorithm splits a secret value into n shares and the $\mathsf{Reconstruct}$ algorithm recovers the initial secret value with only $t + 1$ shares.

We employ in our threshold signature construction the additive secret sharing scheme. In this scheme, a number $x \in \mathbb{Z}_q$ is shared among n parties P_i by sending random $x_i \in \mathbb{Z}_q$ to P_i such that $x = \sum_i^n x_i \mod q$. The reconstruction is achieved by having all parties send their share x_i to each other. Throughout this report, we denote by $[x] := \{[x]_1, \ldots, [x]_n\}$ additive shares in $\mathbb{Z}_q$ and $[\![x]\!]$ additive shares in $\mathbb{Z}_{q-1}$.

2.3 Additively Homomorphic Encryption

Our protocol uses an encryption scheme with the following two homomorphic properties:

- A pair of encrypted ciphertexts can be added.
- An encrypted ciphertext can be multiplied by a plaintext.

Informally, an additively homorphic encryption sheme consists of three algorithms

$$\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}_{\mathsf{pk}}, \mathsf{Dec}_{\mathsf{sk}}),$$

where KeyGen generates the public and private key pair $(\mathsf{sk}, \mathsf{pk})$ and $\mathsf{Enc}_{\mathsf{pk}}$ and $\mathsf{Dec}_{\mathsf{pk}}$ are the encryption and decryption algorithms, respectively. The first homomorphic property guarantees that there exists an efficient operation $\boxplus$ such that $a + b = \mathsf{Dec}_{\mathsf{sk}}\left(\mathsf{Enc}_{\mathsf{pk}}(a) \boxplus \mathsf{Enc}_{\mathsf{pk}}(b)\right)$, the second that there exists an efficient operation $\boxdot$ such that $a \cdot b = \mathsf{Dec}_{\mathsf{sk}}\left(\mathsf{Enc}_{\mathsf{pk}}(a) \boxdot b\right)$. Standard examples of such encryption schemes include those of Paillier [22] and Castagnos-Laguillaumie [7].

2.4 The Underlying MPC Protocol

We make use of the MPC protocol introduced in [27] instantiated with an additive secret share scheme to obtain a non-interactive online phase. The work [27] also introduces the concept of masked factors as follows. A **masked factor** $\langle x \rangle_\lambda$ hiding a non-zero secret $x \in \mathbb{Z}_q^*$ with independent uniformly random mask exponent $\lambda \in \mathbb{Z}_{q-1}$ is given as

$$\langle x \rangle_\lambda = x \cdot h^{-\lambda} \in \mathbb{Z}_q^*,$$

where h is a public generator in $\mathbb{Z}_q^*$. We refer to the element $h^{-\lambda}$ as the multiplicative mask. To simplify notation, we often use $\langle x \rangle$ when λ is implicit. Observe that, in the description of our protocol below, we often include in the random mask exponent, λ_x, its corresponding secret variable label, x. For readability purposes, we might use one or the other.

Protocol. For a description of the protocol we refer to [27, Section 2]. It aims at evaluating a multivariate polynomial of the form:

$$z = \sum_{a=1}^{A} \prod_{m=1}^{M_a} x_{a,m},$$

where $x_{a,m}$ are secrets owned by input parties, a stands for the terms being added and m for the factors being multiplied in one term. The protocol is divided in two phases: a preprocessing phase, ($\mathcal{F}_{\mathsf{PREPROC}}$, [27, Figure 1]), that generates shares $\{[h^{\gamma_a}]\}, \{[\![\lambda_{a,m}]\!]\}$ such that

$$[\![\gamma_a]\!] = \sum_{m=1}^{M_a} [\![\lambda_{a,m}]\!], \tag{1}$$

and a computation phase (π, [27, Figure 2]), that generates the masked factors with $\lambda_{a,m}$ as mask exponents, i.e. $\langle x_{a,m} \rangle_{\lambda_{a,m}}$ and that computes the polynomial in secret shared form using the expression:

$$[z] = \sum_{a=1}^{A} [h^{\gamma_a}] \prod_{m=1}^{M} \langle x_{a,m} \rangle_{\lambda_{a,m}}.$$

The above description assumes the masked factors are computed according to the target polynomial. However, it is possible to modify the aforementioned protocol to address scenarios where certain masked factors are generated independently prior to the MPC protocol execution. In such cases, the associated masked exponents deviate from the constraints set by Expression (1). This deviation occurs notably in the ECDSA signing equation, where the private signing key x undergoes masking during the distributed key generation protocol. To adapt the MPC protocol from [27] to our specific use case, we introduce an additional exponent, denoted as $\lambda_{\mathtt{gap}}$, designed to compensate for the gap in Expression (1). We refer to Section 4.3 for further details.

2.5 Threshold Signature

A threshold signature scheme consists of three efficient algorithms $(\mathsf{DKG}, \mathsf{Sign}, \mathsf{Ver})$ defined as follows. DKG is a distributed key generation protocol which generates the public signing key alongside the signers' shares of the private signing key. Sign is a signing protocol which receives as inputs the message M to be signed alongside the private key shares of each signer. It outputs a valid signature σ which, in turn, can be verified by the Ver algorithm in the same way as the base signature scheme. The proposed scheme follows the informal definition given above adapted to the client-server model.

3 Model and Definitions

Network setup. We assume the network is composed by a client and n signers. The signers also serve as a database storing both encrypted shares of data associated with the secret key and the secret key masked factor itself. We assume the client and the signers are interconnected through an authenticated and synchronous broadcast mechanism.

Threat model. We operate within the adversarial context of most prior works [20, 15, 14, 6, 8, 13]. We assume static corruptions and consider both semi-honest (in the key generation and key agreement protocols) and malicious (in the remaining protocol) adversaries in the dishonest majority setting with abort.

We extend the security model of custodial wallets by allowing for full corruption of signers during the time when the client is online. This includes both, the client preprocessing and the online phases in the threshold signing protocol.

What could go wrong? Threshold signature schemes (TSS) aim to eliminate single points of failure through their distributed nature. Custodial settings assume there is at least one honest signer during the whole process. Our scheme hardens this through the use of secret key shares, allowing for full corruption of signers during the time the client is online. Moreover, the unforgeability security feature is not compromised against user impersonation attacks on the network as the attacker is still lacking the secret key of the encryption method used to decipher the shares. Next, we define TSS in the client-server model.

Definition 3.1 (Threshold signature scheme in the client-server model). *A $(t, n)-$threshold signature scheme in the client-server model allows a client and a set of n signers equipped with an encryption scheme $\mathcal{E}'$ to sign a message M defined by the client, such that any group of $t+1$ signers along with the client can jointly generate a signature, whereas groups lacking the client or having t or fewer signers cannot. The secret key sk of $\mathcal{E}'$ is owned by the client. More specifically, it consists of the following three protocols:*

- DKG *(distributed key generation protocol): every party receives the signing public key y and, additionally, the signers receive encrypted shares of the signing secret key x.*
- $\mathsf{Sign}^{\mathsf{S}}$ *(preprocessing signing protocol): receives the encrypted shares of the signing secret key x from each signer. It outputs a set of preprocessing material.*
- $\mathsf{Sign}^{\mathsf{C}}$ *(signing protocol): receives the message M to be signed from the client, the encrypted shares of the signing secret key x and the preprocessing material from each signer. It outputs a signature σ.*

Signature schemes are defined to be secure under the standard notion of existential unforgeability against chosen message attacks (EU-CMA) introduced in [17]. The generalization of this security notion to the threshold setting was presented for the first time in [16]. We present below an adapted definition of the unforgeability presented in [16] to the client-server model.

Definition 3.2 (Unforgeable threshold signature scheme in the client-server model)**.** *Consider a malicious adversary $\mathcal{A}$ who corrupts at most t signers during DKG and $\mathsf{Sign}^{\mathsf{S}}$ protocol executions and corrupts all n signers during $\mathsf{Sign}^{\mathsf{C}}$ protocol execution. We say that a $(t, n)-$threshold signature scheme in the client-server model $(\mathsf{DKG}, \mathsf{Sign}^{\mathsf{S}}, \mathsf{Sign}^{\mathsf{C}})$ is unforgeable, if no malicious adversary of corruption type $\mathcal{A}$ can produce, with non-negligible probability in the security parameter, the signature on any new message, given the view of the protocol DKG and of the protocols $\mathsf{Sign}^{\mathsf{S}}$ and $\mathsf{Sign}^{\mathsf{C}}$ on input messages $m_1, \ldots, m_k$ which the adversary adaptively chose as well as signatures on those messages.*

4 Threshold ECDSA

In this section, we introduce the fundamental building blocks of our threshold ECDSA scheme, which encompass the distributed key generation (DKG) and signing (Sign) protocols.

Standard primitives. In our approach, we employ the protocol $\pi_{\texttt{PowOpen}}$ over additive shares $[\![x]\!]$, which opens the value g^x to all parties. This is achieved by having all parties broadcasting their component, $g^{[\![x]\!]_i}$, and multiplying all components together. We also use the elliptic curve version of the protocol for shares in $\mathbb{Z}_q$, denoted by $\pi_{\texttt{ECPowOpen}}$. Similarly, given additive shares $[\![x]\!]$ and public generator G, all parties broadcast their component, $[\![x]\!]_i \cdot G$, and add all components together. We incorporate the additive secret sharing version of the joint random generation of secret values, denoted as $\pi_{\texttt{RSS}}$ [16, 23].

4.1 Key Agreement

We present a semi-honest version of the key agreement $\pi_{\texttt{KeyAgreement}}$ protocol (Figure 1) which preserves privacy against malicious adversaries but not correctness. A maliciously secure version can, in principle, be realized using techniques from [18]. It uses a pair of preprocessing material $([\![\lambda]\!], [h^\lambda])$ to compute a pair $(\langle x\rangle_\lambda, y)$, where $\langle x\rangle_\lambda$ is the masked signing secret and y the corresponding public key. Note that h^λ is the inverted mask of x which is hidden under the value y' and x is defined as $x = h^r$, for some random $r \in \mathbb{Z}_{p-1}$. This protocol plays a pivotal role in both DKG and Sign protocols. Notice that this protocol could be replaced by a centralized version where the client locally generates the pair of signing keys and distributes the encrypted shares from the generated private key to the network nodes.

Random masked factor and public elliptic point representation, $\pi_{\texttt{KeyAgreement}}$.

$$(\langle x\rangle_\lambda, y) \leftarrow \pi_{\texttt{KeyAgreement}}([\![\lambda]\!], [h^\lambda])$$

1. Run $[\![r]\!] \leftarrow \pi_{\texttt{RSS}}$.
2. Run in parallel $\langle x\rangle_\lambda := h^r \cdot h^{-\lambda} \leftarrow \pi_{\texttt{PowOpen}}(h, [\![r]\!] - [\![\lambda]\!])$ and $y' := h^\lambda \cdot G \leftarrow \pi_{\texttt{ECPowOpen}}(G, [h^\lambda])$.
3. Locally compute $\langle x\rangle_\lambda \cdot y' = x \cdot G =: y$.

Signers' output: $(\langle x\rangle_\lambda, y)$.

Figure 1: Protocol to generate a random masked factor and the corresponding public elliptic point.

Security. We focus on semi-honest adversaries. Standard protocols $\pi_{\texttt{RSS}}$ (see for instance [3]), $\pi_{\texttt{PowOpen}}$ and $\pi_{\texttt{ECPowOpen}}$ [10] are secure in this setup by standard simulator arguments. We show next that $\langle x\rangle$ and y' only leak y. Indeed, y' hides λ by the discrete logarithm (DL) assumption. Thus, $\langle x\rangle = x \cdot h^{-\lambda}$ masks $x = h^r$ under this assumption. The correctness of step 3 is straightforward. Finally, since r is random, so is the secret key $x = h^r$.

4.2 Key Generation

The distributed key generation protocol, denoted as $\pi_{\texttt{DKG}}$ and presented in Figure 2, is mostly based on the key agreement protocol $\pi_{\texttt{KeyAgreement}}$ introduced in Figure 1. The protocol $\pi_{\texttt{DKG}}$ outputs encrypted signing secret key shares of the form $(\langle x\rangle_\lambda, \mathbf{e}_{\mathsf{pk},\lambda,i})$. Here, $\langle x\rangle_\lambda$ is the masked factor of the secret signing key x and $\mathbf{e}_{\mathsf{pk},\lambda,i} := \mathsf{Enc}_{\mathsf{pk}}([\![\lambda]\!]_i)$ is the encryption of signer i's share of the corresponding masked exponent λ. We can think of $x_i = (\langle x\rangle_\lambda, [\![\lambda]\!]_i)$ as a share of x and the operation $(\langle x\rangle_\lambda, [\![\lambda]\!]_i) \mapsto (\langle x\rangle_\lambda, \mathsf{Enc}_{\mathsf{pk}}([\![\lambda]\!]_i))$ as the encryption method from the scheme $\mathcal{E}'$. This fits the definition of DKG protocol given before for a threshold signature scheme in the client-server model. We denote by $(\langle x\rangle_\lambda, \mathbf{e}_{\mathsf{pk},\lambda})$ the set of all encrypted shares $\{x_i\}$ under $\mathcal{E}'$.

The encryption of the shares is produced as follows. After computing the masked factor of the secret key, the signers encrypt their corresponding mask exponent share, $[\![\lambda_x]\!]$, and subsequently erase the pair $([\![\lambda_x]\!], g^{\lambda_x})$. The encryption scheme must be additively homomorphic for seamless integration with the signing protocol, eliminating the need to decrypt shares during the process. We assume the corresponding key pair was previously generated and distributed accordingly. Its private key, denoted as sk, is securely stored on the client's side, while the corresponding public key, labeled as pk, is disseminated to the entire network.

Performance. The round complexity of $\pi_{\texttt{DKG}}$ protocol aligns with that of the key agreement protocol. The semi-honest version of the protocol involves two rounds of communication and requires one correlation pair.

Security. We present a heuristic argument regarding the security of the $\pi_{\texttt{DKG}}$ protocol. We focus on the semi-honest setting. In accordance with Definition 3.2, we contemplate an adversary $\mathcal{A}$ capable of corrupting all but one (precisely $n-1$) signer. The security of $\pi_{\texttt{DKG}}$ is based on the security of the key agreement protocol and the fact that all parties delete the original pair $([\![\lambda]\!], [h^\lambda])$.

Note that signers lack the ability to reconstruct the private signing key x in case all encrypted signing secret key share tuples $((\langle x\rangle_\lambda, \mathsf{e}_{\mathsf{pk},\lambda}), y)$ are leaked. The security of $\langle x\rangle_\lambda$ relies on the

Distributed Key Generation, $\pi_{\texttt{DKG}}$.

$$\left(\left(\langle x\rangle_\lambda, \mathsf{e}_{\mathsf{pk},\lambda}\right), y\right) \leftarrow \pi_{\texttt{DKG}}()$$

Parameters. $(\mathsf{sk}, \mathsf{pk})$: client's secret and public key.

1. Retrieve a pair $([\![\lambda]\!], [h^\lambda])$ from preprocessed material.
2. Run $(\langle x\rangle_\lambda, y) \leftarrow \pi_{\texttt{KeyAgreement}}([\![\lambda]\!], [h^\lambda])$. Set signing public key as y.
3. Locally encypt $\mathsf{e}_{\mathsf{pk},\lambda} := \mathsf{Enc}_{\mathsf{pk}}([\![\lambda]\!])$ and delete the original pair $([\![\lambda]\!], [h^\lambda])$.

Signers' output: $\left(\left(\langle x\rangle_\lambda, \mathsf{e}_{\mathsf{pk},\lambda}\right), y\right)$**.**

Figure 2: Distributed key generation protocol.

protective mask $h^{-\lambda}$, and λ enjoys security through a homomorphic encryption scheme wherein only the client possesses the corresponding secret key sk. Leveraging the homomorphic property of the encryption scheme, the client together with the signers retain the capability to sign messages without the necessity of revealing the encrypted shares.

4.3 Signature Generation

The threshold signing phase can be divided into three distinct phases: signers preprocessing, client preprocessing, and the online phase. The initial phase constitutes the $\mathsf{Sign}^{\mathsf{S}}$ protocol, while the final two phases together form the $\mathsf{Sign}^{\mathsf{C}}$ protocol within the client-server model definition of threshold signature schemes.

Preliminaries. In the context of ECDSA, the polynomial is given by the expression of the signature term

$$s = k^{-1} \cdot m + k^{-1} \cdot x \cdot r,$$

where $m = H(M)$ is the hash of the message M. For a threshold scheme, one must hide x and k. We instantiate the underlying MPC protocol [27] and apply it to the ECDSA equation above by adding a mask $h^{-\lambda_{\text{gap}}}$ as follows:

$$[s \cdot h^{-\lambda_{\text{gap}}}] = [g^{\gamma_1}] \cdot \langle k^{-1}\rangle_{-\lambda_k} \cdot \langle m\rangle_{\lambda_m+\lambda_{\text{gap}}} + [g^{\gamma_2}] \cdot \langle k^{-1}\rangle_{-\lambda_k} \cdot r \cdot \langle x\rangle_{\lambda_x},$$

where $s \cdot h^{-\lambda_{\text{gap}}} = \langle s\rangle_{\lambda_{\text{gap}}}$ is a masked version of s and Expression 1 translates to

$$\begin{aligned} -[\![\lambda_{\text{gap}}]\!] &= [\![\gamma_1]\!] + [\![\lambda_k]\!] - [\![\lambda_m + \lambda_{\text{gap}}]\!] \\ -[\![\lambda_{\text{gap}}]\!] &= [\![\gamma_2]\!] + [\![\lambda_k]\!] - [\![\lambda_x]\!]. \end{aligned}$$

Notice that the first equation can be rewritten as $[\![\lambda_m]\!] = [\![\gamma_1]\!] + [\![\lambda_k]\!]$.

Protocol. The threshold signing protocol is presented in Figure 3. The $\pi_{\texttt{Sign}}$ protocol begins with the signers preprocessing phase, which aligns with the $\mathsf{Sign}^{\mathsf{S}}$ protocol from Definition 3.1. In this phase, signers use the encrypted shares tuple $(\langle x\rangle_\lambda, \mathsf{e}_{\mathsf{pk},\lambda,i})$ as their input. They start by creating a masked nonce $\langle k\rangle$ and the corresponding public key $k \cdot G$ through a key agreement protocol. After

Threshold signing, $\pi_{\texttt{Sign}}$.

$$\sigma_M = (s, r) \leftarrow \pi_{\texttt{Sign}}(M, (\langle x \rangle, \mathsf{e}_{\mathsf{pk},\lambda_x}))$$

Parameters. $(\mathsf{sk}, \mathsf{pk})$: client's secret and public key.

Signers preprocessing

1. Signers retrieve three pairs from preprocessed material, i.e. $(\llbracket \lambda_k \rrbracket, [h^{\lambda_k}])$, $(\llbracket \gamma_1 \rrbracket, [h^{\gamma_1}])$ and $(\llbracket \gamma_2 \rrbracket, [h^{\gamma_2}])$.
2. Signers generate nonce and the corresponding public value:
$$(\langle k \rangle, k \cdot G) \leftarrow \pi_{\texttt{KeyAgreement}}(\llbracket \lambda_k \rrbracket, [h^{\lambda_k}]).$$
3. Signers locally:
   (a) set $(r, \_) := k \cdot G$.
   (b) compute $\langle k^{-1} \rangle = \langle k \rangle^{-1}$ and set $\llbracket \lambda_{k^{-1}} \rrbracket := -\llbracket \lambda_k \rrbracket$.
4. Signers locally compute encrypted elements:
   (a) $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_m \rrbracket) := \mathsf{Enc}_{\mathsf{pk}}(\llbracket \gamma_1 - \lambda_{k^{-1}} \rrbracket)$.
   (b) $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_{\text{gap}} \rrbracket) := \mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_{k^{-1}} - \gamma_2 \rrbracket) \boxplus \mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_x \rrbracket)$.
   (c) $\mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_1}])$ and $\mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_2}])$.
5. Delete all original pairs $(\llbracket \lambda_k \rrbracket, [h^{\lambda_k}])$, $(\llbracket \gamma_1 \rrbracket, [h^{\gamma_1}])$ and $(\llbracket \gamma_2 \rrbracket, [h^{\gamma_2}])$ and also delete shares $\llbracket \lambda_{k^{-1}} \rrbracket$.

Client preprocessing

6. Signers send $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_{\text{gap}} \rrbracket)$ and $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_m \rrbracket)$ to the client.
7. The client uses sk to decrypt both $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_{\text{gap}} \rrbracket)$ and $\mathsf{Enc}_{\mathsf{pk}}(\llbracket \lambda_m \rrbracket)$. Then, the client reconstructs them and saves $\lambda_m + \lambda_{\text{gap}}$.

Online

11. The client computes $m = H(M)$ and masks it: $\langle m \rangle_{\lambda_m + \lambda_{\text{gap}}} = m \cdot h^{-(\lambda_m + \lambda_{\text{gap}})}$. Sends $\langle m \rangle$ to all signers.
12. Each signer locally computes and sends to the client:
$$\mathsf{Enc}_{\mathsf{pk}}([s \cdot h^{-\lambda_{\text{gap}}}]) = \mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_1}]) \boxdot (\langle k^{-1} \rangle \cdot \langle m \rangle) \boxplus \mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_2}]) \boxdot (\langle k^{-1} \rangle \cdot r \cdot \langle x \rangle) \qquad (2)$$
13. The client uses sk to decrypt the resulting shares, reconstructs the masked signature, multiplies the result by $h^{\lambda_{\text{gap}}}$ and verifies the signature is correct. In case of failure, it aborts.

Signers' output: $\perp$.
Client's output: $\sigma_M = (s, r)$.

Figure 3: Protocol to sign the client's message in a threshold setting.

this initial step, all further actions can be done locally. In step 4, signers encrypt the relevant shares information to ensure resistance against full corruption in the following two phases. The encryption scheme's homomorphic property allows for their effective use in subsequent stages.

Both the client preprocessing and the online phases constitutes the $\mathsf{Sign}^{\mathsf{S}}$ protocol from Definition 3.1. Steps 6 and 7 allow the client to compute the mask exponent $\lambda_m + \lambda_{\text{gap}}$ of the message m. This is introduced for correctness as the signers together are computing encrypted shares of the masked signature.

Performance. The scheme presented uses the MPC protocol in [27] to compute the inverse and multiplication of secrets without communication. These two steps usually make up the most round intensive steps in standard protocols. By separating the work between signers and client preprocessing, the protocol is set up to go through 3 rounds. By combining all the preprocessing steps, we can take it a step further and carry out step 6 at the same time as step 2. It is important to highlight that the elements needed for both steps do not rely on each other. This optimization allows for a 2-round preprocessing protocol, plus the rounds required to generate three preprocessed pairs. Additionally, notice that steps 4(a) and 4(b) can be merged into a single element, which not only saves computational resources but also reduces the bandwidth needed in step 6. We chose to present it this way for the sake of clarity and readability.

Correctness. The correctness follows from: 1) the linear properties of the additively homomorphic encryption scheme, 2) the fact that for a share $[a]$ and masked factor $\langle b \rangle$, $[a] \cdot \langle b \rangle = [a \cdot \langle b \rangle]$ as $\langle b \rangle$ is a public value owned by all signers, and 3) the property $\langle k \rangle_{\lambda_k}^{-1} = \langle k^{-1} \rangle_{-\lambda_k} = \langle k^{-1} \rangle_{\lambda_{k^{-1}}}$, which implies $\lambda_{k^{-1}} := -\lambda_k$. Thus, we have:

$$\begin{aligned} r.h.s &= \mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_1}]) \boxdot (\langle k^{-1} \rangle \cdot \langle m \rangle) \boxplus \mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_2}]) \boxdot (\langle k^{-1} \rangle \cdot r \cdot \langle x \rangle) \\ &= \mathsf{Enc}_{\mathsf{pk}}([h^{\gamma_1} \cdot k^{-1} \cdot h^{-\lambda_{k^{-1}}} \cdot m \cdot h^{-\lambda_m} \cdot h^{-\lambda_{\text{gap}}} + h^{\gamma_2} \cdot k^{-1} \cdot h^{-\lambda_{k^{-1}}} \cdot r \cdot x \cdot h^{-\lambda_x}]) \\ &= \mathsf{Enc}_{\mathsf{pk}}([k^{-1} \cdot m \cdot h^{-\lambda_{\text{gap}}} + k^{-1} \cdot r \cdot x \cdot h^{-\lambda_{\text{gap}}}]) = \mathsf{Enc}_{\mathsf{pk}}([s \cdot h^{-\lambda_{\text{gap}}}]). \end{aligned}$$

Security. Once more, we offer a heuristic argument concerning the security of the $\pi_{\texttt{Sign}}$ protocol, with a detailed security analysis reserved for a subsequent version. In accordance with Definition 3.2, it becomes imperative to contemplate two distinct scenarios: firstly, the corruption of at most $n-1$ signers over the entire protocol; secondly, the compromise of at most $n-1$ signers during the signers' preprocessing phase, coupled with complete corruption occurring during the client preprocessing and online phases.

Firstly, we examine a scenario wherein the adversary corrupts $n-1$ signers throughout the entire protocol execution. The signers' preprocessing phase only involves communication during the key agreement protocol execution. It is noteworthy that the security of step 1 comes from the preprocessing phase of the underlying MPC protocol, while step 2 derives its security from the key agreement protocol. The key agreement protocol, denoted as $\pi_{\texttt{KeyAgreement}}$, achieves semi-honest security, preserving privacy against malicious adversaries but without ensuring correctness. We observe that, in the context of malicious security with abort, it suffices to validate the resulting signature's correctness at the client's end (Step 13) [10].

Moreover, steps 4 and 5 exclusively involve local operations on shares, maintaining resilience against information disclosure to adversaries due to the imposed threshold limit of $n-1$. Steps 6 and 7 provide the client with the mask of the message digest m. It is crucial to highlight that exposing both λ_{gap} and λ_m to the client does not leak information about $[\![\lambda_x]\!]$ and hence about the private key x, as γ_1 and γ_2 mask λ_x, as observed in the expression $\lambda_{\text{gap}} = \gamma_1 - \lambda_m - \gamma_2 + \lambda_x$.

Secondly, the unforgeability of the signature comes from the security of the underlying MPC protocol applied in Expression 2. The $n-1$ signers only have masked elements ($\langle k^{-1}\rangle$, $\langle m\rangle$, $\langle x\rangle$) to either recover x or forge a signature (i.e., compute and reveal the signature SoP) under a different m. This feat remains unattainable without divulging the term masks $[h^{\gamma_1}]$ and $[h^{\gamma_2}]$.

Now, we comment on the security of the scheme even if the adversary is able to corrupt all signers during the client dependent phases (client preprocessing and online). By virtue of encryption and the existence of one honest party up to step 5, we can ensure there is at least one plain-text share missing for every sharing $[\![\lambda_{k^{-1}}]\!]$, $[\![\lambda_m]\!]$, $[\![\lambda_{\text{gap}}]\!]$, $[h^{\gamma_1}]$ and $[h^{\gamma_2}]$. This prevents the adversary from:

- reconstructing λ_{gap}, $\lambda_{k^{-1}}$ and γ_1, which from the linear relation $4(b)$ allows the adversary to recover λ_x and, subsequently, the signing secret key x.
- unmasking k^{-1}, which makes the security of the signing secret key x dependent on the security of the signature element s.
- computing and opening $[k^{-1}\cdot m] = [g^{\gamma_1}]\cdot\langle k^{-1}\rangle_{-\lambda_k}\cdot\langle m\rangle_{\lambda_m+\lambda_{\text{gap}}}$ and $[k^{-1}\cdot r\cdot x] = [g^{\gamma_2}]\cdot\langle k^{-1}\rangle_{-\lambda_k}\cdot r\cdot\langle x\rangle_{\lambda_x}$. This makes the security of the signing secret key x dependent on the security of the message digest m.

After step 5, and even with full node collusion, we note that the signers are not able to forge a signature due to their inability to decrypt in step 7 and also to decrypt the signature in step 12. As a final remark, the adversary is able to deviate from the protocol in step 6. This would allow them to coordinate and send an encryption of elements at their choice. However, this would only provide them with the message digest and they would not even be able to correctly compute Expression 2.

As previously noted, impersonation attacks represent a significant vulnerability within the custodial setting, thereby shifting the security onus onto the authentication mechanism employed by the network. We observe that this scheme is robust against adversaries attempting to bypass the authentication mechanism and successfully impersonate a client. Notably, a fraudulent client would find itself incapable of executing both steps 7 and 12 due to the absence of the required decryption secret key, x.

5 Conclusion

In the context of the dishonest majority setting, we have presented a threshold ECDSA scheme within the client-server model. Our scheme incorporates an efficient preprocessing stage that ensures an optimal online phase involving a single round for the client to transmit a message to the signers and another round to receive the result. Notably, our protocol maintains its unforgeability even in the presence of $n-1$ corruptions and full collusion among signers during the client-dependent phase of the distributed signing protocol. Additionally, it demonstrates resilience against user impersonation attacks, enhancing the overall security of the proposed scheme.

References

[1] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. *A Survey of ECDSA Threshold Signing*. Cryptology ePrint Archive, Paper 2020/1390. https://eprint.iacr.org/2020/1390. 2020.

[2] Jean-Philippe Aumasson and Omer Shlomovits. *Attacking Threshold Wallets*. Cryptology ePrint Archive, Paper 2020/1052. https://eprint.iacr.org/2020/1052. 2020.

[3] J. Bar-Ilan and D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction". In: *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*. PODC '89. Edmonton, Alberta, Canada: Association for Computing Machinery, 1989, pp. 201–209. ISBN: 0897913264.

[4] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography — PKC 2003*. Springer Berlin Heidelberg, Dec. 2002, pp. 31–46.

[5] Luís T.A.N. Brandão, Nicky Mouha, and Apostol Vassilev. *Threshold schemes for cryptographic primitives:* tech. rep. Mar. 2019.

[6] Ran Canetti et al. *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. Cryptology ePrint Archive, Paper 2021/060. https://eprint.iacr.org/2021/060. 2021.

[7] Guilhem Castagnos and Fabien Laguillaumie. "Linearly homomorphic encryption from". In: *Cryptographers' Track at the RSA Conference*. Springer. 2015, pp. 487–505.

[8] Guilhem Castagnos et al. *Bandwidth-efficient threshold EC-DSA*. Cryptology ePrint Archive, Paper 2020/084. https://eprint.iacr.org/2020/084. 2020.

[9] Panagiotis Chatzigiannis et al. *SoK: Web3 Recovery Mechanisms*. Cryptology ePrint Archive, Paper 2023/1575. https://eprint.iacr.org/2023/1575. 2023.

[10] Ivan Damgård et al. "Fast Threshold ECDSA with Honest Majority". In: *Security and Cryptography for Networks*. Springer International Publishing, 2020, pp. 382–400. ISBN: 9783030579906.

[11] Yvo Desmedt. "Society and Group Oriented Cryptography: a New Concept". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1988, pp. 120–127. ISBN: 9783540481843.

[12] Yvo Desmedt and Yair Frankel. "Threshold cryptosystems". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, pp. 307–315.

[13] Adam Gagol et al. *Threshold ECDSA for Decentralized Asset Custody*. Cryptology ePrint Archive, Paper 2020/498. https://eprint.iacr.org/2020/498. 2020.

[14] Rosario Gennaro and Steven Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2018.

[15] Rosario Gennaro and Steven Goldfeder. *One Round Threshold ECDSA with Identifiable Abort*. Cryptology ePrint Archive, Paper 2020/540. https://eprint.iacr.org/2020/540. 2020.

[16] Rosario Gennaro et al. "Robust Threshold DSS Signatures". In: *Advances in Cryptology — EUROCRYPT '96*. Springer Berlin Heidelberg, 1996, pp. 354–371.

[17] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM Journal on Computing* 17.2 (Apr. 1988), pp. 281–308. ISSN: 1095-7111.

[18] Jonathan Katz. *Round Optimal Fully Secure Distributed Key Generation*. Cryptology ePrint Archive, Paper 2023/1094. https://eprint.iacr.org/2023/1094. 2023.

[19] Yehuda Lindell. *Cryptography and MPC in the Coinbase Prime Web3 Wallet*. 2023. URL: https://coinbase.bynder.com/m/68b5c8bedf49a9f9/original/CBPrime-W3W-MPC-White-Paper.pdf.

[20] Yehuda Lindell and Ariel Nof. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2018.

[21] Nillion. "tecdsa". https://github.com/nillion-oss/tecdsa. 2023. Accessed: 2023-11-07.

[22] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Springer Berlin Heidelberg, pp. 223–238.

[23] Torben Pryds Pedersen. "Distributed Provers with Applications to Undeniable Signatures". In: *Advances in Cryptology — EUROCRYPT '91*. Springer Berlin Heidelberg, pp. 221–242.

[24] Joop van de Pol. *Don't overextend your Oblivious Transfer*. 2023. URL: https://blog.trailofbits.com/2023/09/20/dont-overextend-your-oblivious-transfer/ (visited on 10/12/2023).

[25] Alfredo De Santis et al. "How to share a function securely". In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94*. ACM Press, 1994.

[26] Dmytro Tymokhanov and Omer Shlomovits. *Alpha-Rays: Key Extraction Attacks on Threshold ECDSA Implementations*. Cryptology ePrint Archive, Paper 2021/1621. https://eprint.iacr.org/2021/1621. 2021.

[27] Miguel de Vega et al. *Evaluation of Arithmetic Sum-of-Products Expressions in Linear Secret Sharing Schemes with a Non-Interactive Computation Phase*. Cryptology ePrint Archive, Paper 2023/1740. https://eprint.iacr.org/2023/1740. 2023.