# Honest-Majority Threshold ECDSA with Batch Generation of Key-Independent Presignatures[*]

Jonathan Katz    Antoine Urban

**Abstract**

Several protocols have been proposed recently for threshold ECDSA signatures, mostly in the dishonest-majority setting. Yet in so-called *key-management networks*, where a fixed set of servers share a large number of keys on behalf of multiple users, it may be reasonable to assume that a majority of the servers remain uncompromised, and in that case there may be several advantages to using an honest-majority protocol.

With this in mind, we describe an efficient protocol for honest-majority threshold ECDSA supporting *batch generation* of *key-independent* presignatures that allow for "non-interactive" online signing; these properties are not available in existing dishonest-majority protocols. Our protocol offers low latency and high throughput, and runs at an amortized rate of roughly 1.3 ms/presignature (after which signatures can be generated in $\approx$80 $\mu$s).

## 1 Introduction

In a $(t, n)$-threshold signature scheme, a private (signing) key is shared among $n$ servers that can jointly interact to generate a signature; at the same time, an adversary compromising up to $t$ of those servers learns nothing about the key and is unable to generate new signatures. Threshold signature schemes have received a lot of attention in the past few years, primarily (though not only) for their application to the secure management of keys controlling cryptocurrency assets. Because ECDSA is by far the most commonly used signature scheme in that setting, much of this work has focused on protocols for threshold ECDSA, and several "wallet-as-a-service" (WaaS) companies such as Fireblocks, Dfns, and Coinbase are actively using such schemes.

Most recent work on threshold ECDSA has focused on the dishonest-majority setting (where $t = n - 1$), whether for the special case of $n = 2$ [14, 12, 6] or arbitrarily many parties [13, 7, 1, 8]. (We refer to [8, Section 1.1] for a nice survey of that work.) Comparatively less recent attention has focused on the honest majority case [9, 4, 10, 5, 15, 8] where $t < n/2$. Yet in some real-world deployments of threshold ECDSA—particularly in *key-management networks*, where a fixed set of parties (aka servers) share a large number of keys, possibly on behalf of multiple users—it may be reasonable to assume that a majority of the parties remain uncompromised, and there may be significant advantages to using an honest-majority protocol.

In this work we initiate explicit consideration of key-management networks in the context of threshold cryptography, and propose an honest-majority threshold ECDSA protocol especially suited to that setting. In particular, our protocol supports two properties (explained in further

[*]Work done at Dfns Labs. **Email:** `jkatz@dfns.co`/`jkatz2@gmail.com`, `antoine.urban@telecom-paris.fr`

detail in the following section) that are critical for real-world deployments of key-management networks: **key-independent presigning** and **batch presignature generation**. We are not aware of any prior work on threshold ECDSA that offers both these features.

## 1.1 Threshold Cryptography in Key-Management Networks

To the best of our knowledge, all prior work on threshold cryptography only explicitly considers the *monolithic* setting where a single key is shared among $n$ parties. This is in contrast to the *key-management networks* deployed by some WaaS companies, where a fixed set of $n$ servers share a large number of keys, possibly on behalf of multiple users. (For example, Dfns operates a key-management network hosting $\approx$10 million keys.) While it is of course possible to run independent executions of a "monolithic" threshold protocol for each key held by the servers, this misses out on opportunities for efficiency improvements and at the same time ignores some real-world challenges. We discuss two examples next.

*Presignature generation* has become an essential feature of modern threshold signature schemes. Roughly, presignature generation allows parties sharing a key to run some (expensive, interactive) protocol before a message to be signed is known, resulting in shared state (a "presignature") held by the parties. Later, to sign some message, the parties can use an existing presignature to non-interactively generate a signature. (By default, when we refer to presignatures we mean those that allow for non-interactive signing.) Presignature generation is important in many practical deployments of threshold signatures, since it makes the on-line latency for signature generation (which is what the end-user cares about) essentially instantaneous.

Consider using presignature generation in a key-management network where a total of $N$ keys are shared. An immediate problem is that most[1] existing protocols for presignature generation generate *key-dependent* presignatures; that is, each presignature is tied to a specific key shared by the parties. Using such protocols in a key-management network can be done in essentially two ways: the servers can generate one presignature for each key they share (requiring generation and storage of $N$ presignatures, even if many of them may not be used for a long time), or they can selectively generate presignatures for a subset of the shared keys (which risks high latency if they are then asked to issue a signature with a key for which no presignature exists). Neither of these approaches is satisfactory. Thus, for key-management networks it is critical to have *key-independent* presignatures, not tied to any specific key. It is worth noting that there seem to be inherent difficulties in constructing efficient threshold ECDSA protocols with key-independent presignatures in the dishonest-majority setting.[2]

Key-management networks sharing a large number of keys can also be expected to issue signatures at a higher rate than in a monolithic setting. The servers in the network may therefore need to have multiple presignatures available at any time. In that case, it is advantageous for the servers to utilize protocols for *batch* presignature generation that can generate $m \gg 1$ presignatures at an aggregate cost significantly lower than the cost of independently generating $m$ presignatures. (Note that $m$ may be much lower than the number of keys held by the servers.) Such batch presigning is especially beneficial when presignatures are key-independent.

[1] An exception is the work of Damgård et al. [5], which also considers the honest-majority setting. Note, however, that their work does not consider key-management networks and they do not observe the benefits of key-independence.

[2] We do not claim any formal impossibility result, and in fact it *is* possible to construct protocols offering key-independent presignatures in the dishonest-majority setting based on, e.g., fully homomorphic encryption. Such protocols are unlikely to be competitive with state-of-the-art threshold ECDSA protocols, however.

## 1.2 Our Contribution

We focus on designing a threshold ECDSA protocol suitable for deployment in key-management networks. In particular—as discussed in the previous section—we are interested in protocols supporting batch generation of key-independent presignatures. Efficient protocols with these properties seem possible only if an honest majority is assumed and, indeed, this assumption can lead to more-efficient protocols in general; we thus concentrate on the honest-majority setting. Our focus is only on the signing protocol, and we leave distributed key generation out of scope.

Several prior works have designed protocols for threshold ECDSA in the honest-majority setting [9, 4, 10, 5, 15, 8]. Only the protocol of Damgård et al. [5] offers key-independent presignatures; note that aspects of their protocol have been patented (see `https://patents.google.com/patent/US11757657B2/en`). None of the aforementioned works consider batch presigning.

Here we describe a protocol satisfying our requirements, based on techniques for generic secure multiparty computation in the honest-majority setting but adapted to the case of threshold ECDSA. We aim to make our protocol as practical as possible, introducing a number of optimizations and working explicitly in a point-to-point network model (i.e., without assuming broadcast).

**Novelty with respect to prior work.** The core framework for threshold ECDSA that we present in Section 3 has appeared in many prior works, but we are not aware of any prior work that explicitly considers (1) batch presignature generation, or (2) key-independent presignature generation (in a network of signers who hold shares for multiple keys). Techniques for generating multiplication triples (cf. Section 4) have been used many times before, but as we discuss in Section 4 our approach is explicitly directed to batch triple generation and is more efficient than existing approaches targeting general-purpose secure computation. Finally, while pseudorandom secret sharing (cf. Section 5) has been used many times before, we are not aware of any prior work allowing the parties to set up the shared keys themselves (i.e., with no dealer) without any additional additional rounds for commitments or complaint resolution and without a broadcast channel.

**Guide to the paper.** To make the description of our protocols and its proof easier to follow, our construction is entirely *modular*. In Section 3 we give a framework for constructing (honest-majority) threshold ECDSA protocols based on an ideal functionality $\mathcal{F}_{\mathsf{rss}}$ for "random secret sharing" and an ideal functionality $\mathcal{F}_{\mathsf{triple}}$ for generating Beaver (multiplication) triples. In Section 4 we show how to realize $\mathcal{F}_{\mathsf{triple}}$ based on $\mathcal{F}_{\mathsf{rss}}$ and a functionality $\mathcal{F}_{\mathsf{wmult}}$ for "weak" multiplication of shared values. Our protocol for $\mathcal{F}_{\mathsf{triple}}$ involves a batch verification check of (random) multiplication triples based on prior work but optimized for our setting. We realize $\mathcal{F}_{\mathsf{rss}}$ using existing techniques for pseudorandom secret sharing [3] (Section 5), though explicitly adapting that work to a setting without broadcast or trusted setup. (Our protocol realizing $\mathcal{F}_{\mathsf{rss}}$ has complexity exponential in $n$. We target practical deployments in which $n$ is relatively small, e.g., $n < 20$.) We show how to realize $\mathcal{F}_{\mathsf{wmult}}$ (using standard techniques) in Appendix A. Experimental results are included in Section 6.

## 2 Preliminaries

### 2.1 Notation and Background

$\mathbb{G}$ is a group of prime order $q$, with generator $g$. We let $\mathbb{Z}_q$ be the field with $q$ elements, and $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\}$. We let $[n] = \{1, \ldots, n\}$, and $s \leftarrow S$ denote uniform selection of $s$ from finite set $S$.

**Lagrange interpolation.** If $f \in \mathbb{Z}_q[X]$ is a polynomial of degree at most $t$, then it is determined by its values on any $t+1$ distinct points. Thus, for any $j \in \mathbb{Z}_q$ and $S \subset \mathbb{Z}_q$ of size $t+1$, there are efficiently computable *Lagrange coefficients* $\{\lambda^S_{i,j}\}_{i \in S}$ such that $f(j) = \sum_{i \in S} \lambda^S_{i,j} \cdot f(i)$. For any $S \subset \mathbb{Z}_q$ of size $t+1$ and any $\{y_i\}_{i \in S}$ with $y_i \in \mathbb{Z}_q$, we let $\mathsf{interpolate}_t(j, S, \{y_i\}_{i \in S}) = \sum_{i \in S} \lambda^S_{i,j} \cdot y_i$.

When $|S| \geq t+1$, we can verify whether values $\{y_i\}_{i \in S}$ (with $y_i \in \mathbb{Z}_q$) are consistent with a degree-$t$ polynomial $f$ (i.e., whether there exists a polynomial $f$ of degree at most $t$ such that $f(i) = y_i$ for all $i \in S$) by letting $S' \subseteq S$ be an arbitrary subset of size $t+1$ and checking that $y_j \overset{?}{=} \mathsf{interpolate}_t(j, S', \{y_i\}_{i \in S'})$ for all $j \in S \setminus S'$. (Note that when $|S| = t+1$, any values $\{y_i\}_{i \in S}$ are consistent.) Overloading notation, for $|S| \geq t+1$ we let $\mathsf{interpolate}_t(j, S, \{y_i\}_{i \in S})$ be the function that returns $\perp$ if the $\{y_i\}_{i \in S}$ are not consistent with a degree-$t$ polynomial, and otherwise returns $\mathsf{interpolate}_t(j, S', \{y_i\}_{i \in S'})$ for an arbitrary $S' \subseteq S$ of size $t+1$.

We further overload notation by allowing for interpolation "in the exponent." That is, for $S \subset \mathbb{Z}_q$ of size $t+1$ and any $\{g_i\}_{i \in S}$ with $g_i \in \mathbb{G}$, we let $\mathsf{interpolate}_t(j, S, \{g_i\}_{i \in S}) = \prod_{i \in S} g_i^{\lambda^S_{i,j}}$. Note that if we let $x_i = \log_g g_i$ for all $i$, then $\log_g \mathsf{interpolate}_t(j, S, \{g_i\}_{i \in S}) = \mathsf{interpolate}_t(j, S, \{x_i\}_{i \in S})$. For $|S| \geq t+1$, we can verify whether values $\{g_i\}_{i \in S}$ are consistent with a degree-$t$ polynomial in the natural way, and define $\mathsf{interpolate}_t(j, S, \{g_i\}_{i \in S})$ in a manner exactly analogous to above.

**Shamir secret sharing.** The $(t+1)$-out-of-$n$ Shamir secret sharing of a value $x \in \mathbb{Z}_q$ works by setting $a_0 := x$, choosing $a_1, \ldots, a_t \leftarrow \mathbb{Z}_q$, defining the polynomial $f(X) = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}_q[X]$ of degree at most $t$, and outputting the shares $x_1 = f(1), \ldots, x_n = f(n)$. The value of $f$ at any point can be derived from any set of $t+1$ of the shares using Lagrange interpolation; in particular, this allows for reconstructing the secret $x = f(0)$ from any $t+1$ shares.

**ECDSA.** For our purposes, the ECDSA signature scheme works as follows. To sign a hashed message $h = H(\mathsf{msg}) \in \mathbb{Z}_q$ with private key $x \in \mathbb{Z}_q$, the signer chooses $k \leftarrow \mathbb{Z}_q^*$, sets $R := g^k$, and computes $r := F(R) \in \mathbb{Z}_q$ for a publicly known function $F$. It then computes $s := k^{-1} \cdot (h + rx) \bmod q$ and, if $s > q/2$, sets $s := q - s$.[3] It outputs the signature $(r, s)$. Signature $(r, s)$ on hashed message $h$ with respect to public key $y$ is verified by checking that $0 < s < q/2$ and $F(g^{h \cdot s^{-1}} \cdot y^{r \cdot s^{-1}}) = r$. We denote such signature verification by $\mathsf{Vrfy}_y(h, (r, s))$.

## 2.2 Threshold ECDSA

We consider threshold protocols for ECDSA, where private keys are shared by $n$ parties $P_1, \ldots, P_n$, and an adversary who corrupts up to $t$ of those parties should be unable to forge a signature under any key on any message that has not been explicitly signed by the parties. In the honest-majority setting we consider it holds that $t < n/2$. Throughout this work we assume for simplicity that $n = 2t+1$, and that the adversary always corrupts exactly $t$ parties.[4] We let $\mathcal{C} \subset [n]$ denote the indices of the corrupted parties, and let $\mathcal{H} = [n] \setminus \mathcal{C}$ be the indices of the honest parties.

We leave key generation out of scope, and simply assume that the $n$ parties begin holding $(t+1)$-out-of-$n$ Shamir secret shares of one or more private keys $x^{(1)}, \ldots$, with party $P_i$ holding the $i$th share $x_i^{(1)}, \ldots$ of each key. We assume a *coordinator*, distinct from $P_1, \ldots, P_n$, who coordinates execution of the protocol among the $n$ parties, and who holds the (correct) public keys $y^{(1)}, \ldots$ associated with the private keys shared by the parties. The coordinator is assumed to be (semi-)honest; since the

[3]We assume signature normalization is done to prevent malleability attacks. This is not essential for our results.

[4]The protocol of course remains secure if fewer than $t$ parties are corrupted, but some of the ideal functionalities we rely on need to be modified in that case. See footnote 6.

$\mathcal{F}_{\mathsf{ECDSA}}$

**Presigning:** On input $(\mathsf{presign}, m)$ from the coordinator and each of the $n$ parties, do:

1. For $i = 1, \ldots, m$, choose $k_i \leftarrow \mathbb{Z}_q^*$ and compute $R_i := g^{k_i}$ and $r_i := F(R_i)$. Send $R_1, \ldots, R_m$ to the adversary $\mathcal{S}$, who responds with either $\mathsf{abort}$ or $\mathsf{continue}$.
2. If $\mathcal{S}$ sent $\mathsf{abort}$, send $\mathsf{abort}$ to the coordinator. Otherwise, send $\mathsf{completed}$ to the coordinator and store the tuples $\{(k_i, r_i)\}_{i=1}^m$.

**Signing:** On input $(\mathsf{sign}, i, h, y)$ from the coordinator and $(\mathsf{sign}, i, h, x_j)$ from $P_j$ for $j \in \mathcal{H}$, do:

1. Let $x := \mathsf{interpolate}_t(0, \mathcal{H}, \{x_j\}_{j \in \mathcal{H}})$.
2. Compute $s' := k_i^{-1} \cdot (h + r_i \cdot x)$. If $s' > q/2$, set $s := q - s'$; else set $s := s'$.
3. Delete $(k_i, r_i)$. Send $(i, h, y, s')$ to $\mathcal{S}$, who responds with either $\mathsf{abort}$ or $\mathsf{continue}$. If $\mathcal{S}$ sent $\mathsf{abort}$, send $\mathsf{abort}$ to the coordinator. Else, send $(r_i, s)$ to the coordinator.

Figure 1: ECDSA signing functionality.

coordinator in our model determines which messages get signed by the parties, meaningful security is not possible without this assumption.

We assume the parties $P_1, \ldots, P_n$ and the coordinator communicate via a synchronous network in which each pair of parties is connected by a point-to-point secure (i.e., private and authenticated) channel. We do not require a broadcast channel. An adversary can statically corrupt $t$ of the parties, and cause any corrupted party to deviate arbitrarily from the protocol. We assume a *rushing* adversary that can obtain the messages sent by the honest parties in any round of the protocol before corrupted parties send their messages for that round.

Our threshold ECDSA protocols are designed to have a preprocessing phase for batch generation of *presignatures*, following which signing can be done "non-interactively," using one of those presignatures, when a message to be signed is known. The coordinator initiates execution of the different phases of the protocol, and handles usage of the presignatures; see the reactive functionality $\mathcal{F}_{\mathsf{ECDSA}}$ in Figure 1. Each presignature is used only once, and can be used with any key. To initiate computation of $m$ presignatures, the coordinator sends $(\mathsf{presign}, m)$ to each of the $n$ parties. In response, the parties execute a protocol at the end of which (if the execution is not aborted) they each output a collection of $m$ tuples. To initiate computation of a signature on message $\mathsf{msg}$ using the private key associated with public key $y$, the coordinator computes $h = H(\mathsf{msg})$ and sends to the parties an index $i$ indicating which presignature to use, the hashed message $h$, and an indication of which key share to use. (In $\mathcal{F}_{\mathsf{ECDSA}}$ we leave the latter implicit, and instead simply provide each party with their share of the corresponding private key.) The coordinator also tells all parties to delete all information related to the presignature that was used. In response to a signing request, the parties each perform some local computation and send a result back to the coordinator; the coordinator computes and outputs a signature based on the information it receives.

Explicitly, we assume that whenever the parties execute the signing protocol the honest parties each (1) hold the same hashed message $h$, (2) use shares for the private key associated with public key $y$ (thus, their shares form a valid $(t+1)$-out-of-$n$ sharing of $\log_g y$), (3) use the same presignature, and (4) never reuse a presignature. A semi-honest coordinator can enforce all these.

We remark that although unforgeability of ECDSA in a setting where the adversary can observe presignatures before choosing messages to be signed is not equivalent to unforgeability of standard

ECDSA, it has been assumed in prior works and can be shown to hold in the generic group model. We refer to the work of Groth and Shoup [11] for extensive discussion. When the coordinator is (semi-)honest, as we assume, some of the attack scenarios considered in their work do not apply.

We prove that our protocol securely realizes $\mathcal{F}_{\mathsf{ECDSA}}$ for $t$ corrupted parties. Since none of our proofs uses rewinding, one can verify that our proofs carry over to the UC setting.

## 3 A Framework for Honest-Majority Threshold ECDSA

In Figure 2 we give a general framework for constructing protocols for threshold ECDSA realizing $\mathcal{F}_{\mathsf{ECDSA}}$. While our framework is conceptually the same as in several prior works, we stress that we explicitly incorporate batch presigning and key-independent presignatures (in both our ideal functionality and our protocol framework). Our framework is based on ideal functionalities $\mathcal{F}_{\mathsf{rss}}$ (cf. Figure 3) for generating $(t+1)$-out-of-$n$ shares of a random value or $(2t+1)$-out-of-$n$ shares of 0, and $\mathcal{F}_{\mathsf{triple}}$ (cf. Figure 4) for generating shares of two random values and their product. We show how to realize $\mathcal{F}_{\mathsf{rss}}$ based on *pseudorandom secret sharing* in Section 5, and show how to realize $\mathcal{F}_{\mathsf{triple}}$ (based on other ideal functionalities) in Section 4.

**Theorem 1.** *Protocol* $\Pi_{\mathsf{ECDSA}}$ *$t$-securely realizes* $\mathcal{F}_{\mathsf{ECDSA}}$ *in the* $\{\mathcal{F}_{\mathsf{rss}}, \mathcal{F}_{\mathsf{triple}}\}$*-hybrid model.*

Protocol $\Pi_{\mathsf{ECDSA}}$

**Presigning:** On input $(\mathsf{presign}, m)$, each party $P_j$ does:

1. Send $\mathsf{init}$ to $\mathcal{F}_{\mathsf{rss}}$.
2. Call $\mathcal{F}_{\mathsf{rss}}$ on input $(\mathsf{zero}, m)$, and let $\{o_{i,j}\}_{i\in[m]}$ be the result.
3. Call $\mathcal{F}_{\mathsf{triple}}$ on input $(\mathsf{triple}, m)$. If the result is $\mathsf{abort}$ then abort; otherwise, let $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m]}$ be the result.
4. For $i \in [m]$, send $w_{i,j}$ and $R_{i,j} := g^{k_{i,j}}$ to all other parties.
5. Let $w_i := \mathsf{interpolate}_t(0, [n], \{w_{i,j}\}_{j\in[n]})$ and $R_i := \mathsf{interpolate}_t(0, [n], \{R_{i,j}\}_{j\in[n]})$ for all $i \in [m]$. If $w_i \in \{\perp, 0\}$ or $R_i = \perp$ for some $i$, abort. Otherwise, for $i \in [m]$ set $k'_{i,j} := w_i^{-1} \cdot a_{i,j}$ and $r_i := F(R_i)$. Store the tuples $\{(r_i, o_{i,j}, k'_{i,j})\}_{i\in[m]}$ and send $\mathsf{completed}$ to the coordinator.

If the coordinator receives $\mathsf{completed}$ from all parties, it outputs $\mathsf{completed}$; otherwise it outputs $\mathsf{abort}$.

**Signing:** On input $(\mathsf{sign}, i, h, x_j)$, each party $P_j$ does:

1. Set $s_j := k'_{i,j} \cdot (h + r_i \cdot x_j) + o_{i,j}$. Send $(r_i, s_j)$ to the coordinator. Delete $(r_i, o_{i,j}, k'_{i,j})$.

The coordinator, with $(\mathsf{sign}, i, h, y)$, then does:

1. Given $\{(r_j, s_j)\}_{j\in[n]}$, let $r := r_1$. If $r_j \neq r_1$ for some $j$, output $\mathsf{abort}$.
2. Set $s := \mathsf{interpolate}_{2t}(0, [n], \{s_{i,j}\}_{j\in[n]})$; if $s > q/2$, set $s := q - s$. If $\mathsf{Vrfy}_y(h, (r, s)) \neq 1$, output $\mathsf{abort}$; else output $(r, s)$.

Figure 2: General framework for computing ECDSA signatures.

*Proof.* Fix a hybrid-world adversary $\mathcal{A}$ corrupting $\{P_i\}_{i \in \mathcal{C}}$, and set $\mathcal{C}^* := \mathcal{C} \cup \{0\}$. We model the semi-honest coordinator by treating it as honest, but giving $\mathcal{A}$ all the messages it receives. We describe an ideal-world adversary $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{ECDSA}}$ that works as follows:

**Presigning:** When corrupted parties receive $(\mathsf{presign}, m)$, send $(\mathsf{presign}, m)$ to $\mathcal{F}_{\mathsf{ECDSA}}$ on behalf of each corrupted party, and receive $R_1, \ldots, R_m$ in return. Then run $\mathcal{A}$ with the corrupted parties given input $(\mathsf{presign}, m)$ and do:

1. Let $\mathcal{A}$'s inputs to $\mathcal{F}_{\mathsf{rss}}$ be $\{o_{i,j}\}_{i \in [m], j \in \mathcal{C}}$.
2. Let $\mathcal{A}$'s inputs to $\mathcal{F}_{\mathsf{triple}}$ be $\{(a_{i,j}, \bar{k}_{i,j}, \bar{w}_{i,j})\}_{i \in [m], j \in \mathcal{C}}$ and $\mathcal{H}_\perp$. Let $\mathcal{H}_+ := \mathcal{H} \setminus \mathcal{H}_\perp$. For $i \in [m]$ and $j \in \mathcal{C}$, set $\bar{R}_{i,j} := g^{\bar{k}_{i,j}}$.
3. For $i \in [m]$, set $\bar{R}_{i,0} := R_i$. Choose $\bar{w}_{i,0}$ according to[5] the following distribution:
$$\bar{w}_{i,0} = \begin{cases} 0 & \text{with probability } \frac{2}{q} - \frac{1}{q^2} \\ \text{a uniform element of } \mathbb{Z}_q^* & \text{otherwise.} \end{cases}$$
For $i \in [m]$ and $j \in \mathcal{H}_+$, compute $w_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{\bar{w}_{i,\ell}\}_{\ell \in \mathcal{C}^*})$ and $R_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{\bar{R}_{i,\ell}\}_{\ell \in \mathcal{C}^*})$. Give $\mathcal{A}$ the values $\{(w_{i,j}, R_{i,j})\}_{i \in [m], j \in \mathcal{H}_+}$ as the messages sent by the (non-aborting) honest parties. In return, receive for each honest party $P$ the values $\{w_{i,j}^P, R_{i,j}^P\}_{i \in [m], j \in \mathcal{C}}$ sent by $\mathcal{A}$ on behalf of the corrupted parties to $P$.
4. If for any honest $P$ we have $w_{i,j}^P \neq \bar{w}_{i,j}$ or $R_{i,j}^P \neq \bar{R}_{i,j}$ for some $i \in [m]$ and $j \in \mathcal{C}$, or if $\bar{w}_{i,0} = 0$ for some $i \in [m]$, or if $\mathcal{H}_\perp \neq \emptyset$, send $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and stop. Otherwise, give $\mathsf{completed}$ to $\mathcal{A}$ as the message sent by each honest party to the coordinator, and receive from $\mathcal{A}$ the messages from corrupted parties to the coordinator. If any of the corrupted parties fails to send $\mathsf{completed}$ to the coordinator, send $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{ECDSA}}$ and stop.
5. If $\mathsf{abort}$ was not sent to $\mathcal{F}_{\mathsf{ECDSA}}$, send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{ECDSA}}$. Then for $i \in [m]$ do:
   (a) Set $r_i := F(R_i)$.
   (b) For $j \in \mathcal{C}$ set $k'_{i,j} := \bar{w}_{i,0}^{-1} \cdot a_{i,j}$.
   Store the tuples $\{(r_i, \{(o_{i,j}, k'_{i,j})\}_{j \in \mathcal{C}})\}_{i \in [m]}$.

**Signing:** When each corrupted $P_j$ receives $(\mathsf{sign}, i, h, x_j)$, forward those to $\mathcal{F}_{\mathsf{ECDSA}}$ and receive $s'$ in return. Let $\mathcal{C}^{**}$ be a set of size $2t+1$ with $\mathcal{C}^* \subset \mathcal{C}^{**} \subset [n] \cup \{0\}$, set $r := r_i$, and do:

1. Set $s_0 := s'$. For $j \in \mathcal{C}$, set $s_j := k'_{i,j} \cdot (h + r \cdot x_j) + o_{i,j}$. For $j \in \mathcal{C}^{**} \setminus \mathcal{C}^*$, choose $s_j \leftarrow \mathbb{Z}_q$. For $j \in \mathcal{H} \setminus \mathcal{C}^{**}$, set $s_j := \mathsf{interpolate}_{2t}(j, \mathcal{C}^{**}, \{s_\ell\}_{\ell \in \mathcal{C}^{**}})$.
2. Run $\mathcal{A}$ with corrupted parties given their respective inputs, and with $\{(r, s_j)\}_{j \in \mathcal{H}}$ as the messages sent by honest parties to the coordinator. Receive the messages $\{(r_j, \bar{s}_j)\}_{j \in \mathcal{C}}$ sent by corrupted parties to the coordinator.
3. For $j \in \mathcal{H}$, set $\bar{s}_j := s_j$. Set $\bar{s} := \mathsf{interpolate}_{2t}(0, [n], \{\bar{s}_j\}_{j \in [n]})$. If $r_j \neq r$ for some $j \in \mathcal{C}$, or if $\bar{s} \neq \pm s'$, or if $s' = 0$, send $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{ECDSA}}$. Otherwise, send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{ECDSA}}$.

When $\mathcal{A}$'s execution is complete, output whatever $\mathcal{A}$ outputs.

[5]This is the distribution of the product of two uniform elements of $\mathbb{Z}_q$.

$\mathcal{F}_{\mathsf{rss}}$

Let $\mathcal{C}^* := \mathcal{C} \cup \{0\}$. Let $j^*$ be the lowest index in $\mathcal{H}$, and let $\mathcal{H}^* := \mathcal{H} \setminus \{j^*\}$.

Init: On input init from all parties in $\mathcal{H}$, send initialized to all parties. Each of the following can then be called at most once.

Rand: On input $(\mathsf{rand}, m)$ from each party in $\mathcal{H}$ and $\{r_{i,j}\}_{i\in[m],j\in\mathcal{C}}$ from the adversary, do:

1. For $i \in [m]$, choose $r_{i,0} \leftarrow \mathbb{Z}_q$.
2. For $i \in [m]$ and $j \in \mathcal{H}$, set $r_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{r_{i,\ell}\}_{\ell\in\mathcal{C}^*})$.
3. For $j \in \mathcal{H}$, send $\{r_{i,j}\}_{i\in[m]}$ to $P_j$.

Zero: On input $(\mathsf{zero}, m)$ from each party in $\mathcal{H}$ and $\{o_{i,j}\}_{i\in[m],j\in\mathcal{C}}$ from the adversary, do:

1. For $i \in [m]$, set $o_{i,0} := 0$. For $i \in [m]$ and $j \in \mathcal{H}^*$, choose $o_{i,j} \leftarrow \mathbb{Z}_q$.
2. For $i \in [m]$, set $o_{i,j^*} := \mathsf{interpolate}_{2t}(j^*, \mathcal{H}^* \cup \mathcal{C}^*, \{o_{i,j}\}_{j\in\mathcal{H}^*\cup\mathcal{C}^*})$.
3. For $j \in \mathcal{H}$, send $\{o_{i,j}\}_{i\in[m]}$ to $P_j$.

Figure 3: Ideal functionality for "random secret sharing."

$\mathcal{F}_{\mathsf{triple}}$

Let $\mathcal{C}^* := \mathcal{C} \cup \{0\}$.

1. Receive $(\mathsf{triple}, m)$ from each honest party, and $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m],j\in\mathcal{C}}$ and $\mathcal{H}_\perp \subseteq \mathcal{H}$ from the adversary.
2. For $i \in [m]$ do:
   (a) Choose $a_{i,0}, k_{i,0} \leftarrow \mathbb{Z}_q$. Set $w_{i,0} := a_{i,0} \cdot k_{i,0}$.
   (b) For $j \in \mathcal{H}$, compute the shares $a_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{a_{i,\ell}\}_{\ell\in\mathcal{C}^*})$, $k_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{k_{i,\ell}\}_{\ell\in\mathcal{C}^*})$, and $w_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{w_{i,\ell}\}_{\ell\in\mathcal{C}^*})$.
3. For $j \in \mathcal{H}$ do: if $j \in \mathcal{H}_\perp$ send abort to $P_j$; else send $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m]}$ to $P_j$.

Figure 4: Ideal functionality for batch generation of multiplication triples.

We claim that the distribution of the output of $\mathcal{A}$ and the outputs of the coordinator in the hybrid world is statistically indistinguishable from the distribution of the output of $\mathcal{S}$ and the outputs of the coordinator in the ideal world. In fact, if we let Bad denote the event that $R_i = 1$ for some $i$ (which cannot occur in the ideal world, and occurs with negligible probability in the hybrid world), and condition on the event that Bad does not occur (which we denote by $\overline{\mathsf{Bad}}$), the distributions are *identical*. To see this, compare the above execution of $\mathcal{S}$ in the ideal world to an execution of $\mathcal{A}$ with $\Pi^{t,n}_{\mathsf{ECDSA}}$ in the hybrid world conditioned on $\overline{\mathsf{Bad}}$:

**Presigning:** We first consider the presigning phase.

1. The view of $\mathcal{A}$ in its interaction with ideal functionalities $\mathcal{F}_{\mathsf{rss}}$ and $\mathcal{F}_{\mathsf{triple}}$ in the hybrid world is the same as in the execution of $\mathcal{A}$ as a subroutine of $\mathcal{S}$ in the ideal world.
2. The shares $\{k_{i,j}\}_{i\in[m],j\in\mathcal{H}_+}$ sent to the honest parties by $\mathcal{F}_{\mathsf{triple}}$ in the hybrid world are uniquely determined by the inputs sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{triple}}$ and the uniform values $\{k_{i,0}\}_{i\in[m]}$

chosen internally by $\mathcal{F}_{\mathsf{triple}}$. Thus, the values $\{R_{i,j}\}_{i\in[m],j\in\mathcal{H}_+}$ are uniquely determined by the inputs sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{triple}}$ and the values $\{R_i = g^{k_{i,0}}\}_{i\in[m]}$. The distribution of the $\{R_i\}_{i\in[m]}$ chosen by $\mathcal{F}_{\mathsf{ECDSA}}$ in the ideal world is identical to the distribution of those values in the hybrid world conditioned on $\overline{\mathsf{Bad}}$. Thus, the distribution of the $\{R_{i,j}\}_{i\in[m],j\in\mathcal{H}_+}$ given to $\mathcal{A}$ as a subroutine of $\mathcal{S}$ in the ideal world is identical to the distribution of those messages in the hybrid world (conditioned on $\overline{\mathsf{Bad}}$).

A similar argument applies to $\{w_{i,j}\}_{i\in[m],j\in\mathcal{H}_+}$, even conditioned on $\{R_{i,j}\}_{i\in[m],j\in\mathcal{H}_+}$.

3. Assuming $\mathcal{H}_\perp = \emptyset$, the messages $\{(w_{i,j}, R_{i,j})\}_{i\in[m],j\in\mathcal{H}}$ sent by the honest parties in the hybrid world uniquely determine values $\{(w_i, R_i)\}_{i\in[m]}$ as well as the messages that are supposed to be sent by the corrupted parties. Thus, in particular, any deviation by the corrupted parties in the messages they send would cause the honest parties (and hence the coordinator) to abort in either the hybrid or ideal worlds.
4. Assuming the honest parties have not aborted in the hybrid world, they will all send $\mathsf{completed}$ to the coordinator. So the coordinator's output depends only on whether the corrupted parties all send $\mathsf{completed}$ to the coordinator or not. The view of the coordinator in the hybrid world is thus identically distributed to the simulated view of the coordinator in the ideal world, and the output of the coordinator is also identical in the hybrid and ideal worlds.

Assuming successful completion of the presigning phase in the hybrid world, let $k'_{i,j} := w_i^{-1} \cdot a_{i,j}$ for $i \in [m]$ and $j \in \mathcal{C}$. For $i \in [m]$, the values $\{k'_{i,j}\}_{j\in[n]}$ are $(t+1)$-out-of-$n$ shares of $k_i^{-1}$, where $k_i = \log_g R_i$. The honest parties also hold shares $\{o_{i,j}\}_{i\in[m],j\in\mathcal{H}}$ that are uniform and independent subject to the constraint that the $\{o_{i,j}\}_{i\in[m],j\in[n]}$ are $(2t+1)$-out-of-$n$ shares of 0.

**Signing:** Now consider an execution of the signing phase in the hybrid world, where each corrupted party $P_j$ has input $(\mathsf{sign}, i, h, x_j)$. The input values $\{x_j\}_{j\in[n]}$ (including those being used by the honest parties) are, by assumption, $(t+1)$-out-of-$n$ shares of a private key $x$ corresponding to the public key $y$ held by the coordinator, and the shares $\{x_j\}_{j\in\mathcal{H}}$ of the honest parties uniquely determine $x$ as well as the shares $\{x_j\}_{j\in\mathcal{C}}$ held by the corrupted parties. Letting $s_j := k'_{i,j} \cdot (h + r_i \cdot x_j) + o_{i,j}$ for $j \in \mathcal{C}$, it follows that the $\{s_j\}_{j\in\mathcal{H}}$ sent by the honest parties are uniform subject to the constraint that the $\{s_j\}_{j\in[n]}$ are $(2t+1)$-out-of-$n$ shares of $s' = k_i^{-1} \cdot (h + r_i \cdot x)$. This matches the distribution of the $\{s_j\}_{j\in\mathcal{H}}$ in the ideal world.

$\mathcal{A}$ can either cause the coordinator to abort or to output a correct signature (on hashed message $h$ with respect to public key $y = g^x$) with first component $r_i$. Since the first component of a valid signature uniquely determines the second component, this implies that the output of the coordinator in the hybrid world matches what would be output in the ideal world.

This completes the proof. $\square$

**Running multiple executions.** As written, $\Pi_{\mathsf{ECDSA}}$ requires parties to re-initialize $\mathcal{F}_{\mathsf{rss}}$ each time the presigning phase is run. This, in turn, is necessary because the $\mathcal{F}_{\mathsf{rss}}$ functionality only allows a single call to each of $\mathsf{Rand}/\mathsf{Zero}$ per initialization. Looking ahead to the protocol realizing $\mathcal{F}_{\mathsf{rss}}$ (cf. Section 5), these limitations are due to the fact that the protocol involves calls to a pseudorandom function that requires appropriate domain separation. In practice, such domain separation can be enforced within an execution of the protocol by using distinct identifiers, and across executions by incorporating a non-repeating session-id; this allows initialization of $\mathcal{F}_{\mathsf{rss}}$ to be done once-and-for-all, after which an unbounded number of invocations of presigning/signing can be done.

## 4 Secure Triple Generation

In this section, we show an approach for securely realizing $\mathcal{F}_{\mathsf{triple}}$. Our construction is inspired by the work of Chida et al. [2]. In their work, roughly speaking, parties multiply secret-shared values in the course of evaluating a circuit using a *weak* multiplication functionality $\mathcal{F}_{\mathsf{wmult}}$ that preserves privacy of inputs and outputs but allows the adversary to introduce arbitrary additive shifts in the results. (See Figure 5.) The parties additionally share a random value $r$, and for each multiplication of shared values $a, b$ to give output $ab$ they also compute shares of $ra, rb$, and $rab$ using the same weak multiplication functionality. At the end of the computation of the circuit, the parties perform a probabilistic check using fresh random values to verify that the adversary has not introduced an additive shift in any of the multiplications.

The technique of Chida et al. [2], which is geared to general secure computation, does not directly support preprocessing because the values being multiplied depend on the inputs to the circuit being evaluated. In the context of ECDSA, however, it suffices to securely generate multiplication triples during a preprocessing phase, as shown in Section 3. (Fundamentally, this is because in the context of ECDSA correctness of the output can be verified using the public key.) Directly using the technique of Chida et al. to evaluate a circuit that outputs $m$ multiplication triples would involve generating $3m + 1$ random values and performing $4m$ weak multiplications. We observe that it is enough to generate $2m + 2$ random values and perform $3m$ weak multiplications. Roughly, this is because we do not need to preserve privacy of any of the shared values when cheating is detected.

In Figure 6 we show a protocol $\Pi_{\mathsf{triple}}$ for realizing $\mathcal{F}_{\mathsf{triple}}$ based on $\mathcal{F}_{\mathsf{wmult}}$. (We show how to realize $\mathcal{F}_{\mathsf{wmult}}$ using standard techniques in Appendix A.) Before turning to the full proof of security for protocol $\Pi_{\mathsf{triple}}$, we provide some intuition. In the protocol, parties first generate shares of uniform values $\{a_i\}_{i\in[m]}$, $\{k_i\}_{i\in[m]}$, and $r, \beta$, where $a_i = \mathsf{interpolate}_t(0, \mathcal{H}, \{a_{i,j}\}_{j\in\mathcal{H}})$ and $k_i$, $r$, and $\beta$ are defined similarly. They then use $\mathcal{F}_{\mathsf{wmult}}$ to compute shares of $\{w_i\}_{i\in[m]}$ (where $w_i$ is

$\mathcal{F}_{\mathsf{wmult}}$

Let $\mathcal{C}^* := \mathcal{C} \cup \{0\}$.

1. Receive $\{(a_{i,j}, k_{i,j})\}_{i\in[m], j\in\mathcal{H}}$ from the honest parties. (If some honest parties $\mathcal{H}' \subseteq \mathcal{H}$ do not provide input, then send $\mathsf{abort}$ to all honest $P_j$, send $(\mathsf{abort}, \mathcal{H}')$ to the adversary, and halt.) For $i \in [m]$ and $j \in \mathcal{C}^*$, set $a_{i,j} := \mathsf{interpolate}_t(j, \mathcal{H}, \{a_{i,\ell}\}_{\ell\in\mathcal{H}})$ and $k_{i,j} := \mathsf{interpolate}_t(j, \mathcal{H}, \{k_{i,\ell}\}_{\ell\in\mathcal{H}})$. Send $\{(a_{i,j}, k_{i,j})\}_{i\in[m], j\in\mathcal{C}}$ to the adversary.
2. Receive $\left(\{d_i\}_{i\in[m]}, \{w_{i,j}\}_{i\in[m], j\in\mathcal{C}}\right)$ from the adversary.
3. For $i \in [m]$ do:
   (a) Set $w_{i,0} := a_{i,0} \cdot k_{i,0} + d_i$.
   (b) For $j \in \mathcal{H}$, set $w_{i,j} := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{w_{i,\ell}\}_{\ell\in\mathcal{C}^*})$.
4. For $j \in \mathcal{H}$, send $\{w_{i,j}\}_{i\in[m]}$ to $P_j$.

Figure 5: Functionality for (weak) multiplication secure up to additive attacks.

Protocol $\Pi_{\mathsf{triple}}$

On input $(\mathsf{triple}, m)$, each party $P_j$ does:

1. Send $\mathsf{init}$ to $\mathcal{F}_{\mathsf{rss}}$.
2. Call $\mathcal{F}_{\mathsf{rss}}$ on input $(\mathsf{rand}, 2m+2)$. Denote the first $2m$ results by $\{a_{i,j}\}_{i\in[m]}$ and $\{k_{i,j}\}_{i\in[m]}$, and the final two results by $r_j, \beta_j$.
3. Call $\mathcal{F}_{\mathsf{wmult}}$ with inputs $\{(k_{i,j}, a_{i,j})\}_{i\in[m]}$ and $\{(r_j, a_{i,j})\}_{i\in[m]}$, and let $\{w_{i,j}\}_{i\in[m]}$ and $\{\mu_{i,j}\}_{i\in[m]}$, respectively, be the results.
4. Call $\mathcal{F}_{\mathsf{wmult}}$ with inputs $\{(\mu_{i,j}, k_{i,j})\}_{i\in[m]}$, and let $\{\tau_{i,j}\}_{i\in[m]}$ be the result.
5. Send $r_j, \beta_j$ to all parties. Let $r := \mathsf{interpolate}_t(0, [n], \{r_j\}_{j\in[n]})$ and $\beta := \mathsf{interpolate}_t(0, [n], \{\beta_j\}_{j\in[n]})$. If $r = \perp$ or $\beta = \perp$, abort.
6. Compute $T_j = \sum_{i=1}^{m}(\tau_{i,j} - r \cdot w_{i,j}) \cdot \beta^i$ and send it to all parties. Let $T := \mathsf{interpolate}_t(0, [n], \{T_j\}_{j\in[n]})$. If $T \neq 0$, abort; otherwise, output $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m]}$.

Figure 6: Realizing $\mathcal{F}_{\mathsf{triple}}$ in the $\{\mathcal{F}_{\mathsf{rss}}, \mathcal{F}_{\mathsf{wmult}}\}$-hybrid model.

supposed to equal $k_i \cdot a_i$), $\{\mu_i\}_{i\in[m]}$ (where $\mu_i$ is supposed to equal $r \cdot a_i$), and $\{\tau_i\}_{i\in[m]}$ (where $\tau_i$ is supposed to equal $\mu_i \cdot k_i$). Finally, they reconstruct $r$ and $\beta$, and publicly reveal

$$T = \sum_{i=1}^{m}(\tau_i - rw_i) \cdot \beta^i.$$

If all parties behave honestly, then $\tau_i = ra_i \cdot k_i = rw_i$ for all $i$ and so $T = 0$.

The more interesting case is when the adversary exploits the weak multiplication functionality to give incorrect output by using a nonzero shift. The following lemma shows that such behavior is detected by the honest users with overwhelming probability.

**Lemma 1.** *Let $w_i = k_i a_i + d_i$, $\mu_i = ra_i + \delta_i$, and $\tau_i = \mu_i k_i + \delta'_i$. If there exists an $i^*$ such that $d_{i^*}, \delta_{i^*}$, or $\delta'_{i^*}$ is nonzero, then $T \neq 0$ except with probability at most $(m+1)/q$.*

*Proof.* The shifts $\{d_i\}_{i\in[m]}, \{\delta_i\}_{i\in[m]}$, and $\{\delta'_i\}_{i\in[m]}$ are all controlled by the adversary, but are independent of $\{k_i\}_{i\in[m]}, r$, and $\beta$. We have

$$\begin{aligned} T &= \sum_{i=1}^{m}(\tau_i - rw_i) \cdot \beta^i \\ &= \sum_{i=1}^{m}\big(\mu_i k_i + \delta'_i - r \cdot (k_i a_i + d_i)\big) \cdot \beta^i \\ &= \sum_{i=1}^{m}\big(k_i \cdot (ra_i + \delta_i) + \delta'_i - r \cdot (k_1 a_i + d_i)\big) \cdot \beta^i \\ &= \sum_{i=1}^{m}\big(k_i\delta_i + \delta'_i - rd_i\big) \cdot \beta^i \; = \; \sum_{i=1}^{m} T_i \cdot \beta^i, \end{aligned}$$

where $T_i \overset{\text{def}}{=} k_i\delta_i + \delta'_i - rd_i$. We now consider different cases:

1. If there is an $i^*$ with $d_{i^*} \neq 0$, then—since $r$ is uniform and independent of everything else—it holds that $T_{i^*} \neq 0$ except with probability $1/q$.

2. If $d_i = 0$ for all $i$ but there is an $i^*$ with $\delta_{i^*} \neq 0$, then—since $k_{i^*}$ is uniform and independent of everything else—it holds that $T_{i^*} \neq 0$ except with probability $1/q$.

3. If $d_i = \delta_i = 0$ for all $i$ but there is an $i^*$ with $\delta'_{i^*} \neq 0$, then $T_{i^*} = \delta'_{i^*} \neq 0$.

Thus, if there exists an $i^*$ such that $d_{i^*}, \delta_{i^*}$, or $\delta'_{i^*}$ is nonzero, then $T(X) \overset{\text{def}}{=} \sum_{i=1}^{m} T_i \cdot X^i$ is a nonzero polynomial of degree at most $m$ except with probability at most $1/q$; assuming that to be the case, $T = T(\beta) \neq 0$ except with probability at most $m/q$. This completes the proof. □

We remark that if $T \neq 0$ the adversary may potentially learn information about $\{(a_{i,j}, k_{i,j})\}_{i\in[m]}$. However, since honest parties abort when $T \neq 0$ and the $\{(a_{i,j}, k_{i,j})\}_{i\in[m]}$ are just random values (that do not need to be kept private), this is not a problem in our setting.

**Theorem 2.** *Protocol $\Pi_{\mathsf{triple}}$ $t$-securely realizes $\mathcal{F}_{\mathsf{triple}}$ in the $\{\mathcal{F}_{\mathsf{rss}}, \mathcal{F}_{\mathsf{wmult}}\}$-hybrid model.*

*Proof.* Fix some hybrid-world adversary $\mathcal{A}$. We describe an adversary $\mathcal{S}$ operating in the ideal world with access to $\mathcal{F}_{\mathsf{triple}}$. When corrupted parties receive input $(\mathsf{triple}, m)$, adversary $\mathcal{S}$ does:

1. Run $\mathcal{A}$ with the corrupted parties given input $(\mathsf{triple}, m)$. Let $\{(a_{i,j}, k_{i,j})\}_{i\in[m],j\in\mathcal{C}}$, $\{r_j\}_{j\in\mathcal{C}}$, and $\{\beta_j\}_{j\in\mathcal{C}}$ be the inputs $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{rss}}$.

   Choose $r, \beta \leftarrow \mathbb{Z}_q$, and set $r_0 := r$ and $\beta_0 := \beta$. For $j \in \mathcal{H}$, set $r_j := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{r_\ell\}_{\ell\in\mathcal{C}^*})$ and $\beta_j := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{\beta_\ell\}_{\ell\in\mathcal{C}^*})$.

2. Send $\{(k_{i,j}, a_{i,j})\}_{i\in[m],j\in\mathcal{C}}$, and $\{r_j\}_{j\in\mathcal{C}}$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{wmult}}$. Let the inputs $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{wmult}}$ be $(\{d_i\}_{i\in[m]}, \{w_{i,j}\}_{i\in[m],j\in\mathcal{C}})$ and $(\{\delta_i\}_{i\in[m]}, \{\mu_{i,j}\}_{i\in[m],j\in\mathcal{C}})$.

3. Send $\{(\mu_{i,j}, k_{i,j})\}_{i\in[m],j\in\mathcal{C}}$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{wmult}}$, and let $(\{\delta'_i\}_{i\in[m]}, \{\tau_{i,j}\}_{i\in[m],j\in\mathcal{C}})$ be the inputs $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{wmult}}$.

4. Let $\mathcal{H}_+ := \mathcal{H}$. For each honest party $P$ receive the values $\{(r_j^P, \beta_j^P)\}_{j\in\mathcal{C}}$ sent by $\mathcal{A}$ on behalf of the corrupted parties to $P$. For each honest $P$ for which $r_j^P \neq r_j$ or $\beta_j^P \neq \beta_j$ for some $j \in \mathcal{C}$, remove $P$ from $\mathcal{H}_+$.

5. If $d_i = \delta_i = \delta'_i = 0$ for all $i$, then:

   (a) For $j \in \mathcal{C}$, compute $T_j := \sum_{i=1}^{m}(\tau_{i,j} - r \cdot w_{i,j}) \cdot \beta^i$. Set $T_0 := 0$. For $j \in \mathcal{H}_+$, set $T_j := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{T_\ell\}_{\ell\in\mathcal{C}^*})$. Send $\{T_j\}_{j\in\mathcal{H}_+}$ to $\mathcal{A}$ on behalf of the parties in $\mathcal{H}_+$. In return, receive for each honest party $P$ values $\{T_j^P\}_{j\in\mathcal{C}}$ sent by $\mathcal{A}$.

   (b) For each $P \in \mathcal{H}_+$ for which $T_j^P \neq T_j$ for some $j \in \mathcal{C}$, remove $P$ from $\mathcal{H}_+$. Then send $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m],j\in\mathcal{C}}$ and $\mathcal{H}_\perp := \mathcal{H} \setminus \mathcal{H}_+$ to $\mathcal{F}_{\mathsf{triple}}$.

6. If $d_i \neq 0$, $\delta_i \neq 0$, or $\delta'_i \neq 0$ for some $i \in [m]$, do:

   (a) For $i \in [m]$, choose $a_{i,0}, k_{i,0} \leftarrow \mathbb{Z}_q$ and set $w_{i,0} := a_{i,0} \cdot k_{i,0} + d_i$, $\mu_{i,0} := r \cdot a_{i,0} + \delta_i$, and $\tau_{i,0} := \mu_{i,0} \cdot k_{i,0} + \delta'_i$. Set $T_0 := \sum_{i=1}^{m}(\tau_{i,0} - r \cdot w_{i,0}) \cdot \beta^i$. If $T_0 = 0$ output $\mathsf{fail}$ and stop. Otherwise, for $j \in \mathcal{C}$, compute $T_j := \sum_{i=1}^{m}(\tau_{i,j} - r \cdot w_{i,j}) \cdot \beta^i$, and for $j \in \mathcal{H}_+$, set $T_j := \mathsf{interpolate}_t(j, \mathcal{C}^*, \{T_\ell\}_{\ell\in\mathcal{C}^*})$.

(b) Send $\{T_j\}_{j\in\mathcal{H}_+}$ to $\mathcal{A}$. Then send $\{(a_{i,j}, k_{i,j}, w_{i,j})\}_{i\in[m],j\in\mathcal{C}}$ and $\mathcal{H}_\perp := \mathcal{H}$ to $\mathcal{F}_{\mathsf{triple}}$.

7. Output whatever $\mathcal{A}$ outputs and stop.

The statistical difference between the distribution of the output of $\mathcal{A}$ and the outputs of the honest parties in the hybrid world and the distribution of the output of $\mathcal{S}$ and the outputs of the honest parties in the ideal world is bounded by the probability with which $\mathcal{S}$ outputs $\mathsf{fail}$ in the ideal world. By Lemma 1, this is at most $(m+1)/q$. □

## 5 Pseudorandom Secret Sharing

To realize $\mathcal{F}_{\mathsf{rss}}$, we rely on *pseudorandom secret sharing* (PRSS) [3] which we now describe. For $t \le n$, let $\mathbb{S}_{n-t,n}$ denote the collection of all subsets of $[n]$ of size $n-t$. For $A \in \mathbb{S}_{n-t,n}$, let $f_A \in \mathbb{Z}_q[X]$ be the polynomial of degree at most $t$ such that

$$f_A(x) = \begin{cases} 1 & x = 0 \\ 0 & x \in [n] \setminus A. \end{cases}$$

Let $\Psi : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{Z}_q$ be a pseudorandom function with key length $\kappa$. Assume there are keys $\{k_A\}_{A\in\mathbb{S}_{n-t,n}}$ such that each party $P_i$ holds $\{k_A\}_{i\in A}$. Then parties can non-interactively generate a $(t+1)$-out-of-$n$ sharing of a secret indexed by $\alpha$ by having each $P_i$ compute the share

$$\sigma_i^\alpha := \sum_{A\in\mathbb{S}_{n-t,n} \,:\, i\in A} \Psi_{k_A}(\alpha)\cdot f_A(i).$$

To see that this gives a valid $(t+1)$-out-of-$n$ Shamir sharing, define the polynomial

$$\alpha(X) = \sum_{A\in\mathbb{S}_{n-t,n}} \Psi_{k_A}(\alpha)\cdot f_A(X)$$

that has degree at most $t$. Then observe that for $i \in [n]$ it holds that

$$\alpha(i) = \sum_{A\in\mathbb{S}_{n-t,n}} \Psi_{k_A}(\alpha)\cdot f_A(i) = \sum_{A\in\mathbb{S}_{n-t,n} \,:\, i\in A} \Psi_{k_A}(\alpha)\cdot f_A(i) = \sigma_i^\alpha.$$

The value defined by these shares is

$$\alpha(0) = \sum_{A\in\mathbb{S}_{n-t,n}} \Psi_{k_A}(\alpha)\cdot f_A(0) = \sum_{A\in\mathbb{S}_{n-t,n}} \Psi_{k_A}(\alpha).$$

If $k_{\mathcal{H}}$ is uniform and independent of the other keys, then for any set $\mathcal{C}$ of $t$ corrupted parties and any distinct values $\alpha_1,\ldots$, the shared values $\alpha_1(0),\ldots$ are jointly pseudorandom, even conditioned on the keys held by the corrupted parties. This holds regardless of how the $\{k_A\}_{A\ne\mathcal{H}}$ are chosen (since $k_{\mathcal{H}}$ is independent of the other keys).

The above can be extended to generate a random $(2t+1)$-out-of-$n$ sharing of 0, something referred to as *pseudorandom zero sharing* (PRZS). Assume keys $\{k_A\}_{A\in\mathbb{S}_{n-t,n}}$ distributed as before. Now, a party $P_i$ can compute a 0-sharing indexed by $\beta$ as

$$\rho_i^\beta = \sum_{\substack{A\in\mathbb{S}_{n-t,n}\\ i\in A}} \sum_{j=1}^{t} \Psi_{k_A}(\beta\|j)\cdot i^j\cdot f_A(i)$$

(where "$\|$" denotes concatenation). These shares correspond to points on the polynomial

$$\beta(X) = \sum_{A \in \mathbb{S}_{n-t,n}} \sum_{j=1}^{t} \Psi_{k_A}(\beta \| j) \cdot X^j \cdot f_A(X),$$

which has degree at most $2t$ and satisfies $\beta(0) = 0$. If $k_{\mathcal{H}}$ is uniform and independent of the other keys as above, it can be verified that (even given the view of an adversary corrupting up to $t$ parties) the shares $\{\rho_i^\beta\}_{i \in \mathcal{H}}$ are uniform subject to the constraint that $\mathsf{interpolate}_{2t}(0, [n], \{\sigma_i^\beta\}_{i \in [n]}) = 0$.

In prior work on PRSS/PRZS, it is assumed that a trusted dealer chooses keys and distributes them to the appropriate parties, or else a complex protocol is run to securely establish those keys. We observe that neither of these are necessary, and it suffices to have a designated party $P_A^*$ for each set $A \in \mathbb{S}_{n-t,n}$ (say, $P_A^* = P_i$ where $i$ is the smallest index in $A$) choose a uniform key $k_A \in \{0,1\}^\kappa$ and send it (via private channel) to each $P_i$ with $i \in A$. At a high level, this is still secure since

- The key $k_{\mathcal{H}}$ will still be chosen uniformly and independently of the other keys, shared correctly among the honest parties, and unknown to the adversary.
- For any set $A \in \mathbb{S}_{n-t,n}$ that contains a corrupted party, the adversary anyway learns $k_A$ even when a trusted dealer distributes keys.
- If the designated party for some subset $A$ is corrupted, that party can send inconsistent keys to different parties in $A$. Nevertheless, for PRSS the shares computed by the honest parties still lie on[6] a degree-$t$ polynomial and define a uniform secret; for PRZS, the shares computed by the honest parties are jointly uniform subject to the linear constraint above (that depends on the view of the corrupted parties), as required by the functionality.

The overall protocol is described in Figure 7.

**Theorem 3.** *If $\Psi$ is a pseudorandom function, then protocol* $\mathsf{PRSS}$ *$t$-securely realizes $\mathcal{F}_{\mathsf{rss}}$.*

*Proof.* Fix some adversary $\mathcal{A}$ attacking $\mathsf{PRSS}$ and corrupting the set of parties $\mathcal{C}$. Consider the ideal-world adversary $\mathcal{S}$ interacting with $\mathcal{F}_{\mathsf{rss}}$ that operates as follows:

$\mathsf{Init}$: On input $\mathsf{init}$ to parties in $\mathcal{C}$, for all $A \in \mathbb{S}_{n-t,n}$ do:

1. If $A = \mathcal{H}$ then do nothing.
2. If $A \neq \mathcal{H}$ and $P_A^*$ is honest, then choose $k_A \leftarrow \{0,1\}^\kappa$ and send it to $\mathcal{A}$. For $j \in A \cap \mathcal{H}$, set $k_A^j := k_A$.
3. If $P_A^*$ is corrupt, then receive $\{k_A^j\}_{j \in A \cap \mathcal{H}}$ from $\mathcal{A}$.

$\mathsf{Rand}$: On input $(\mathsf{rand}, m)$ to parties in $\mathcal{C}$, do:

1. For $j \in \mathcal{H}$ and $i \in [m]$, compute $r'_{i,j} := \sum_{A \in \mathbb{S}_{n-t,n} \setminus \mathcal{H} : j \in A} \Psi_{k_A^j}(0 \| i) \cdot f_A(j)$.
2. For $j \in \mathcal{C}$ and $i \in [m]$, compute $r'_{i,j} := \mathsf{interpolate}_t(j, \mathcal{H}, \{r_{i,\ell}\}_{\ell \in \mathcal{H}})$.
3. Send $\{r'_{i,j}\}_{i \in [m], j \in \mathcal{C}}$ to $\mathcal{F}_{\mathsf{rss}}$.

[6]This assumes exactly $t+1$ honest parties. If there are more than $t+1$ honest parties then the definition of $\mathcal{F}_{\mathsf{rss}}$ needs to be modified appropriately.

Protocol PRSS

Init: For every set $A \in \mathbb{S}_{n-t,n}$ do:

1. Let $P^*_A$ be a designated party with $P^*_A \in \{P_i\}_{i\in A}$.
2. $P^*_A$ chooses $k_A \leftarrow \{0,1\}^\kappa$ and sends $k_A$ to $\{P_i\}_{i\in A}$.
3. Each party $P_j$ lets $k^j_A$ be the key it received from $P^*_A$ for set $A$.

Rand: On input $(\mathsf{rand}, m)$, each party $P_j$ does:

1. For $i \in [m]$, set $r_{i,j} := \sum_{A\in\mathbb{S}_{n-t,n}\,:\,j\in A} \Psi_{k^j_A}(0\|i) \cdot f_A(j)$.
2. Output $\{r_{i,j}\}_{i\in[m]}$.

Zero: On input $(\mathsf{zero}, m)$, each party $P_j$ does:

1. For $i \in [m]$, set $o_{i,j} := \sum_{A\in\mathbb{S}_{n-t,n}\,:\,j\in A} \sum_{\ell=1}^{t} \Psi_{k^j_A}(1\|i\|\ell) \cdot j^\ell \cdot f_A(j)$.
2. Output $\{o_{i,j}\}_{i\in[m]}$.

Figure 7: Protocol for pseudorandom secret sharing.

Zero: On input $(\mathsf{zero}, m)$ to parties in $\mathcal{C}$, do:

1. For $j \in \mathcal{H}$ and $i \in [m]$, compute $o_{i,j} := \sum_{\substack{A\in\mathbb{S}_{n-t,n}\setminus\mathcal{H}\\ j\in A}} \sum_{\ell=1}^{t} \Psi_{k^j_A}(1\|i\|\ell) \cdot j^\ell \cdot f_A(j)$.
2. Let $j^*$ be the smallest index in $\mathcal{C}$, and let $\overline{\mathcal{C}} := (\mathcal{C} \setminus \{j^*\}) \cup \{0\}$.
3. For $j \in \overline{\mathcal{C}}$ and $i \in [m]$, set $o_{i,j} := 0$.
4. For $i \in [m]$, set $o_{i,j^*} := \mathsf{interpolate}_{2t}(j^*, \overline{\mathcal{C}} \cup \mathcal{H}, \{o_{i,\ell}\}_{\ell\in\overline{\mathcal{C}}\cup\mathcal{H}})$.
5. Send $\{o_{i,j}\}_{i\in[m],j\in\mathcal{C}}$ to $\mathcal{F}_{\mathsf{rss}}$.

When $\mathcal{A}$'s execution is complete, output whatever $\mathcal{A}$ outputs.

It is clear that the view (and hence the output) of $\mathcal{A}$ in the real world is identically distributed to its view (and hence the output of $\mathcal{S}$) in the ideal world. Since no messages are exchanged following initialization, we focus on the outputs of the honest parties. We show that the distributions of their outputs in the real and ideal worlds are indistinguishable. For simplicity and notational clarity, we assume $m = 1$ and so omit reference to the index $i$.

**Invocation of Rand.** Consider a real-world execution of Rand. Each honest party $P_j$ outputs

$$\begin{aligned} r_j = \sum_{A\in\mathbb{S}_{n-t,n}\,:\,j\in A} \Psi_{k^j_A}(0) \cdot f_A(j) &= \Psi_{k_\mathcal{H}}(0) \cdot f_\mathcal{H}(j) + \sum_{A\in\mathbb{S}_{n-t,n}\setminus\mathcal{H}\,:\,j\in A} \Psi_{k^j_A}(0) \cdot f_A(j) \\ &= \Psi_{k_\mathcal{H}}(0) \cdot f_\mathcal{H}(j) + r'_j, \end{aligned}$$

where $r'_j \stackrel{\text{def}}{=} \sum_{A\in\mathbb{S}_{n-t,n}\setminus\mathcal{H}\,:\,j\in A} \Psi_{k^j_A}(0) \cdot f_A(j)$.

Consider next an experiment $\mathsf{Expt}_1$ where we choose $r \leftarrow \mathbb{Z}_q$ and then set $r_j := r \cdot f_\mathcal{H}(j) + r'_j$ for all $j \in \mathcal{H}$. Since $\Psi$ is a pseudorandom function, the distribution of the outputs of the honest parties in $\mathsf{Expt}_1$ is computationally indistinguishable from the distribution of their outputs in the real-world execution.

Let $f'$ be the polynomial of degree at most $t$ with $f'(j) = r'_j$ for $j \in \mathcal{H}$. In experiment $\mathsf{Expt}'_1$, we choose $r \leftarrow \mathbb{Z}_q$ and then let $f$ be the polynomial of degree at most $t$ with $f(0) = r$ and $f(j) = 0$ for $j \in \mathcal{C}$. Finally, set $r_j := f(j) + f'(j)$ for all $j \in \mathcal{H}$. It is easily seen that $\mathsf{Expt}'_1$ is merely an alternate, but equivalent, way of describing $\mathsf{Expt}_1$.

$\mathsf{Expt}_2$ proceeds as in $\mathsf{Expt}'_1$, except that we now let $f$ be the polynomial of degree at most $t$ with $f(0) = r - f'(0)$ and $f(j) = 0$ for $j \in \mathcal{C}$. Since $f(0)$ is uniformly distributed in both experiments, the honest parties' outputs are identically distributed in $\mathsf{Expt}_2$ and $\mathsf{Expt}'_1$.

Let $r'_j \overset{\text{def}}{=} \mathsf{interpolate}_t(j, \mathcal{H}, \{r'_\ell\}_{\ell\in\mathcal{H}})$ for $j \in \mathcal{C}$, and let $f'' = f + f'$. Observe that $f''$ has degree at most $t$ and satisfies $f''(0) = r$ and $f''(j) = r'_j$ for $j \in \mathcal{C}$.

Consider the following experiment $\mathsf{Expt}_3$. Set $r'_j := \mathsf{interpolate}_t(j, \mathcal{H}, \{r'_\ell\}_{\ell\in\mathcal{H}})$ for $j \in \mathcal{C}$. Choose uniform $r \leftarrow \mathbb{Z}_q$, and let $f''$ be the polynomial of degree at most $t$ with $f''(0) = r$ and $f''(j) = r'_j$ for $j \in \mathcal{C}$. Finally, set $r_j := f''(j)$ for $j \in \mathcal{H}$. This is an alternate, but equivalent, way of describing $\mathsf{Expt}_2$. But it is also equivalent to the ideal-world execution involving the adversary $\mathcal{S}$ described earlier, and thus completes the proof for $\mathsf{Rand}$.

**Invocation of $\mathsf{Zero}$.** Consider a real-world execution of $\mathsf{Zero}$ with $\mathcal{A}$. Honest party $P_j$ outputs

$$\begin{aligned} o_j &= \sum_{\ell=1}^{t} \Psi_{k_\mathcal{H}}(1\|\ell) \cdot j^\ell \cdot f_\mathcal{H}(j) + \sum_{\substack{A\in\mathbb{S}_{n-t,n}\setminus\mathcal{H} \\ j\in A}} \sum_{\ell=1}^{t} \Psi_{k_A^j}(1\|\ell) \cdot j^\ell \cdot f_A(j) \\ &= \sum_{\ell=1}^{t} \Psi_{k_\mathcal{H}}(1\|\ell) \cdot j^\ell \cdot f_\mathcal{H}(j) + o'_j, \end{aligned}$$

where $o'_j \overset{\text{def}}{=} \sum_{\substack{A\in\mathbb{S}_{n-t,n}\setminus\mathcal{H} \\ j\in A}} \sum_{\ell=1}^{t} \Psi_{k_A^j}(1\|\ell) \cdot j^\ell \cdot f_A(j)$.

Consider next an experiment $\mathsf{Expt}_1$ where we instead choose $r_1, \ldots, r_t \leftarrow \mathbb{Z}_q$ and then set $o_j := \sum_{\ell=1}^{t} r_\ell \cdot j^\ell \cdot f_\mathcal{H}(j) + o'_j$ for $j \in \mathcal{H}$. The fact that $\Psi$ is a pseudorandom function means that the distribution of the outputs of the honest parties in $\mathsf{Expt}_1$ is computationally indistinguishable from the distribution of their outputs in the real-world execution.

Let $j^*$ be the smallest index in $\mathcal{C}$, and let $\overline{\mathcal{C}} = (\mathcal{C} \setminus \{j^*\}) \cup \{0\}$. Let $f'$ be the polynomial of degree at most $2t$ with $f'(j) = 0$ for $j \in \overline{\mathcal{C}}$ and $f'(j) = o'_j$ for $j \in \mathcal{H}$. In experiment $\mathsf{Expt}'_1$ we proceed as follows. Choose $r_1, \ldots, r_t \leftarrow \mathbb{Z}_q$ and let $f(X) = \sum_{\ell=1}^{t} r_\ell \cdot X^\ell \cdot f_\mathcal{H}(X)$. Then set $o_j := f(j) + f'(j)$ for $j \in \mathcal{H}$. This is merely an alternate, but equivalent, way of expressing $\mathsf{Expt}_1$.

Let $j^{**}$ be the smallest index in $\mathcal{H}$, and let $\mathcal{H}^* := \mathcal{H} \setminus \{j^{**}\}$. In $\mathsf{Expt}_2$ we choose $r_j \leftarrow \mathbb{Z}_q$ for $j \in \mathcal{H}^*$ and then let $f$ be the polynomial of degree at most $2t$ such that $f(j) = 0$ for $j \in \mathcal{C} \cup \{0\}$ and $f(j) = r_j$ for $j \in \mathcal{H}^*$. Polynomial $f'$ is defined as before. Then set $o_j := f(j) + f'(j)$ for $j \in \mathcal{H}$. The distributions of the honest parties' outputs in $\mathsf{Expt}_2$ and $\mathsf{Expt}'_1$ are identical (the proof follows from that of PRZS [3]).

In $\mathsf{Expt}_3$ we choose $r_j \leftarrow \mathbb{Z}_q$ for $j \in \mathcal{H}^*$ and then let $f$ be the polynomial of degree at most $2t$ such that $f(j) = 0$ for $j \in \mathcal{C} \cup \{0\}$ and $f(j) = r_j - f'(j)$ for $j \in \mathcal{H}^*$. The $\{o_j\}_{j\in\mathcal{H}}$ are then computed as before. Since the $\{f(j)\}_{j\in\mathcal{H}^*}$ are uniformly distributed in both $\mathsf{Expt}_2$ and $\mathsf{Expt}_3$, this does not change the distribution of the honest parties' outputs.

Let $f'' = f + f'$, and observe that $f''$ has degree at most $2t$ with

$$f''(j) = \begin{cases} 0 & j \in \overline{\mathcal{C}} \\ f'(j^*) & j = j^* \\ r_j & j \in \mathcal{H}^*. \end{cases} \tag{1}$$

In $\mathsf{Expt}_4$, choose $r_j \leftarrow \mathbb{Z}_q$ for $j \in \mathcal{H}^*$ and let $f''$ be the polynomial of degree at most $2t$ satisfying the constraints of Eq. (1); then set $o_j := f''(j)$ for $j \in \mathcal{H}$. This is an alternate, but equivalent, way of describing $\mathsf{Expt}_3$. But it is also equivalent to the ideal-world execution involving $\mathcal{S}$, and thus completes the proof for $\mathsf{Zero}$. $\square$

## 6 Experimental Results

We compare the performance of our protocol to the CGGMP protocol [1], a leading protocol for threshold ECDSA in the dishonest-majority setting. Both protocols were implemented in Rust using the same underlying cryptographic libraries. We then evaluated both protocols for $n = 5$, $t = 2$ by simulating multiple parties executing each protocol (with a coordinator) on a single M1 Macbook Pro with artificial network delays. Both protocols allow for presigning after which signature generation is network-bounded; i.e., signature generation requires only $\approx 80$ $\mu$s of local computation, so the time required to generate a signature is equal to the time required for the parties to communicate with the coordinator. We therefore focus on presigning.

| Network delay | Batch size | CGGMP | Here |
|---|---|---|---|
| 0 | 1 | 1074 | 1.27 |
| 50 | 1 | 1500 | 680 |
|  | 10,000 | — | 1.30 |

Table 1: Amortized time for presignature generation. All times in ms.

Table 1 shows the amortized time for each protocol to generate one presignature, as a function of the network delay and the batch size $m$. (Note that CGGMP does not support batch presignature generation.) Unsurprisingly, given the threat models for which the two protocols were designed, our protocol is noticeably faster than the CGGMP protocol even without batching for network delay up to 50 ms. The effect of batching becomes significant in the presence of network delay. (When there is no network delay, the amortized time to generate a presignature in our protocol is almost unchanged.) Specifically, using a batch size of $m = 10,000$ reduces the (amortized) time to generate a presignature by over $500\times$.

## Acknowledgments

We thank Nikita Sorokovikov and Denis Varlakov for implementing our protocol and running the experiments in Section 6, and thank Orestis Alpos and Chelsea Komlo for helpful feedback.

## References

[1] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *ACM Conf. on Computer and Communications Security (CCS)*, pages 1769–1787. ACM, 2020.

[2] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. *J. Cryptology*, 36(3), 2023. Preliminary version in Crypto 2018.

[3] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference (TCC) 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, 2005.

[4] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS 2020*, volume 12309 of *LNCS*, pages 654–673. Springer, 2020.

[5] Ivan Damgård, Thomas P. Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. *J. Comp. Security*, 30(1):167–196, 2022.

[6] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *IEEE Symp. on Security and Privacy*, pages 980–997. IEEE, 2018.

[7] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE, 2019.

[8] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *IEEE Symp. on Security and Privacy 2023*. IEEE, 2023.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. *Information and Computarion*, 164(1):54–84, 2001. Preliminary version in Eurocrypt '99.

[10] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody, 2020. Available at `https://eprint.iacr.org/2020/498`.

[11] Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. In *Advances in Cryptology—Eurocrypt 2022, Part I*, volume 13275 of *LNCS*, pages 365–396. Springer, 2022.

[12] Yehuda Lindell. Fast secure two-party ECDSA signing. *J. Cryptology*, 34(4), 2021. Preliminary version in Crypto 2017.

[13] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM Conf. on Computer and Communications Security (CCS)*, pages 1837–1854. ACM, 2018.

[14] Philip D. MacKenzie and Michael K. Reiter. Two-party generation of DSA signatures. *Intl. J. Info. Security*, 2(3-4):218–239, 2004. Preliminary version in Crypto 2001.

[15] Michaella Pettit. Efficient threshold-optimal ECDSA. In *CANS 2021*, volume 13099 of *LNCS*, pages 116–135. Springer, 2021.

## A Realizing Weak Multiplication

Here we show how to realize $\mathcal{F}_{\mathsf{wmult}}$ based on $\mathcal{F}_{\mathsf{rss}}$. Although the protocol is standard, we are not aware of any previous analysis of the protocol when a broadcast channel is not available.

Protocol $\Pi_{\mathsf{wmult}}$

On input $\{(a_{i,j}, k_{i,j})\}_{i\in[m]}$, each party $P_j$ does:

1. Send $\mathsf{init}$ to $\mathcal{F}_{\mathsf{rss}}$.
2. Call $\mathcal{F}_{\mathsf{rss}}$ on input $(\mathsf{rand}, m)$. Denote the result by $\{r_{i,j}\}_{i\in[m]}$.
3. Call $\mathcal{F}_{\mathsf{rss}}$ on input $(\mathsf{zero}, m)$. Denote the result by $\{o_{i,j}\}_{i\in[m]}$.
4. For $i \in [m]$ compute $e_{i,j} := a_{i,j} \cdot k_{i,j} + r_{i,j} + o_{i,j}$ and send $e_{i,j}$ to all parties. If some party does not send a value, abort.
5. For $i \in [m]$, compute $e_i := \mathsf{interpolate}_{2t}(0, [n], \{e_{i,j}\}_{j\in[n]})$ and output $\{w_{i,j} := e_i - r_{i,j}\}_{i\in[m]}$.

Figure 8: Realizing $\mathcal{F}_{\mathsf{wmult}}$ in the $\mathcal{F}_{\mathsf{rss}}$-hybrid model.

**Theorem 4.** *Protocol $\Pi_{\mathsf{wmult}}$ $t$-securely realizes $\mathcal{F}_{\mathsf{wmult}}$ in the $\mathcal{F}_{\mathsf{rss}}$-hybrid model.*

*Proof.* For simplicity we assume $m = 1$ and omit the indexing by $i$. Fix some hybrid-world adversary $\mathcal{A}$. We describe an adversary $\mathcal{S}$ operating in the ideal world with access to $\mathcal{F}_{\mathsf{wmult}}$. Upon receiving input for the corrupted parties, $\mathcal{S}$ does:

1. Run $\mathcal{A}$ with corrupted parties given their inputs. Let $\{r_j\}_{j\in\mathcal{C}}$ be the inputs $\mathcal{A}$ sends to the first invocation of $\mathcal{F}_{\mathsf{rss}}$, and let $\{o_j\}_{j\in\mathcal{C}}$ be the inputs $\mathcal{A}$ sends to the second invocation of $\mathcal{F}_{\mathsf{rss}}$.
2. Let $\{(a_j, k_j)\}_{j\in\mathcal{C}}$ be the values received from $\mathcal{F}_{\mathsf{wmult}}$. For $j \in \mathcal{H}$, choose $e_j \leftarrow \mathbb{Z}_q$ and send $\{e_j\}_{j\in\mathcal{H}}$ to $\mathcal{A}$ on behalf of the honest parties. For $j \in \mathcal{C}$, set $\bar{e}_j := a_j \cdot k_j + r_j + o_j$. For $j \in \mathcal{H}$, set $\bar{e}_j := e_j$. Let $\bar{e} := \mathsf{interpolate}_{2t}(0, [n], \{\bar{e}_j\}_{j\in[n]})$.
3. Let $\mathcal{H}_\perp := \emptyset$. Then for $j \in \mathcal{H}$ do:
   - Let $\{e_\ell\}_{\ell\in\mathcal{C}}$ be the values sent to $P_j$ by $\mathcal{A}$ on behalf of the corrupted parties.
   - Compute $e := \mathsf{interpolate}_{2t}(0, [n], \{e_\ell\}_{\ell\in[n]})$ and set $d_j := e - \bar{e}$.

   Finally, set $d := \mathsf{interpolate}_t(0, \mathcal{H}, \{d_j\}_{j\in\mathcal{H}})$.
4. For $j \in \mathcal{C}$, set $w_j := \bar{e} - r_j$. Send $(d, \{w_j\}_{j\in\mathcal{C}})$ to $\mathcal{F}_{\mathsf{wmult}}$ and output whatever $\mathcal{A}$ outputs.

One can verify that the distributions of the outputs of $\mathcal{A}$ and the honest parties in the hybrid and ideal worlds are equivalent. □