Threshold ECDSA in Three Rounds

Jack Doerner
j@ckdoerner.net
Technion

Yashvanth Kondi
yash@ykondi.net
Aarhus University

Eysa Lee
lee.ey@northeastern.edu
Northeastern University

abhi shelat
abhi@neu.edu
Northeastern University

May 26, 2023

Abstract

We present a three-round protocol for threshold ECDSA signing with malicious security against a dishonest majority, which information-theoretically UC-realizes a standard threshold signing functionality, assuming ideal commitment and two-party multiplication primitives. Our work improves upon and fully subsumes the DKLs t-of-n [DKLs19] and 2-of-n [DKLs18] protocols. This document focuses on providing a succinct but complete description of the protocol and its security proof, and contains little expository text.

Contents

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is among the most common and widely deployed cryptographic tools of any kind. Since its standardization by the US National Institute of Standards and Technology (NIST) [Nat13], it has become an ubiquitous component of the internet infrastructure. This makes it a natural and essential target for threshold cryptography. Unfortunately, ECDSA features a non-linear signing equation that is challenging to compute in a distributed fashion.

In this work, we propose a three-round protocol for ECDSA signing that information-theoretically UC-realizes a standard threshold ECDSA signing functionality against a malicious adversary corrupting a dishonest majority of parties, assuming ideal commitments and ideal secure multiplication. Specifically, we prove the theorem:

Theorem 1.1 (Informal Security Theorem)**.** *In the* $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Zero}}, \mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{DLKeyGen}})$*-hybrid model,* $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *statistically UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *against a malicious adversary that statically corrupts up to* $t-1$ *parties.*

where n is the number of parties in total, t is the threshold of parties required for a signature to be produced, and $\mathcal{G}$ is the description of an elliptic curve. In addition to the commitment functionality $\mathcal{F}_{\mathsf{Com}}$ and the randomized multiplication functionality $\mathcal{F}_{\mathsf{RVOLE}}$, our protocol uses $\mathcal{F}_{\mathsf{Zero}}$ to generate secret-sharings of zero, and we abstract the key generation process behind the $\mathcal{F}_{\mathsf{DLKeyGen}}$ functionality.

Our protocol reduces the round complexity of threshold ECDSA signing to parity with threshold signing for the Schnorr scheme [Sch89], for which an elegant three-round protocol has long been known under standard assumptions [Lin22].[1] We derive our protocol primarily from the t-of-n [DKLs19] and 2-of-n [DKLs18] signing schemes of Doerner et al. and combine their approach to achieving malicious security by checking consistency of intermediate computations in the signing curve with a protocol structure recently introduced by Abram et al. [ANO+22] and also used by Groth and Shoup [GS22a] in the contexts of PCG-based signing and honest-majority signing, respectively.[2] To our knowledge, no other threshold ECDSA scheme for an arbitrary threshold requires fewer than four rounds without some flavor of pipelining and preprocessing. The round count of our protocol can be reduced further to two rounds via pipelining, and in the two-party context under pipelining only a single message in each direction is necessary. We suggest to realize the required secure multiplication functionality via a refinement of the protocol of Doerner et al. [DKLs19], and show via closed-form analysis that with this realization, the bandwidth cost of our protocol is significantly reduced, relative to its progenitors. Thus, the protocol in this work fully subsumes the prior works of Doerner et al. [DKLs18, DKLs19] as well as several other threshold ECDSA schemes.

[1] A number of two-round distributed Schorr protocols such as MuSig2 [NRS21] and FROST [BCK+22] also exist, with game-based security under non-standard assumptions.

[2] It seems that the same correlation was also used by Lindell and Nof [LN18] somewhat earlier, without being formulated explicitly.

The purpose of this document is to provide a functional description of our protocol and a proof of its security. We assume familiarity with the problem space, and do not include a comprehensive literature survey, design rationale, or other introductory material. In addition to our primary dishonest-majority protocol, we also give a brief account of a significantly simplified two-round honest-majority protocol.

1.1 Background

ECDSA uses a basic discrete logarithm key pair comprising a uniform $\mathsf{sk} \leftarrow \mathbb{Z}_q$ and a public $\mathsf{pk} = \mathsf{sk} \cdot G$, where $\mathcal{G} = (\mathbb{G}, G, q)$ is an elliptic curve. A signature consists of a public nonce $R = r \cdot G$, where r is a secret per-signature *instance key*, and a value of the form $s = (a + \mathsf{sk} \cdot b)/r$, where a and b are public coefficients. It is the computation of s that forms the core challenge of distributing the signing process efficiently.

One widely-used approach involves sampling secret shares of r, then using the inversion protocol of Bar-Ilan and Beaver [BB89] to compute secret shares of $1/r$, and then a further secure multiplication subprotocol to compute shares of sk/r, before finally assembling the signature. The sequentiality of this approach is its shortcoming: each of the subprotocols requires multiple rounds, and still more rounds may be necessary to ensure security against malicious adversary.

In a recent work on threshold ECDSA in the Pseudorandom Correlation Generation (PCG) paradigm, Abram et al. [ANO+22] proposed to *fuse* two instances of the Bar-Ilan and Beaver protocol to compute secret sharings of what they refer to as an *ECDSA tuple*, which comprises four values (ϕ, r, u, v) such that $u = \phi \cdot r$ and $v = \phi \cdot \mathsf{sk}$. It is easy to see that given these four values, if $w = a \cdot \phi + b \cdot v$ and u are made public, then a valid signature can easily be assembled as w/u. If the secret sharing scheme is linear, then shares of w can be computed locally by the signers, and assembly of the signature takes only the one round required to swap shares of w and u.

Fixing the above correlation as the layout of a putative protocol leaves two problems: how to compute the correlation securely, and, if the adversary is malicious, how to ensure that the correlation is consistent with pk and R, and that it is not altered during assembly. Prior works involving the same correlation have used honest-majority techniques [GS22a], complex PCG techniques with separate statistical MACs [ANO+22], or expensive zero-knowledge proofs [LN18] to solve these problems. In this work, we propose a simple pairwise statistical consistency check, similar in spirit to the global consistency checks used by the protocol of Doerner et al. [DKLs19], which works in the dishonest-majority setting, leverages the structure of the existing shares, requires *no additional* correlated state or zero-knowledge proofs to be generated. We also propose to sample the correlation with a simple and minimal pairwise random vector oblivious linear evaluation (VOLE) functionality—essentially a flavor of secure multiplication—and show how a two-round protocol to realize this functionality efficiently from Oblivious Transfer (OT) can be derived from the secure multiplication protocol of Doerner et al. [DKLs19]. While we focus on our suggested

VOLE realization, the modularity and simplicity of our ECDSA signing protocol imply that its performance properties can be adjusted to mimic or exceed those of threshold ECDSA schemes based on Paillier [Pai99, CGG+20] or class groups [CL15, CCL+23] simply by replacing the underlying VOLE.

2 Preliminaries

2.1 Notation

We use $=$ for equality, $:=$ for right-to-left assignment, $=:$ for left-to-right assignment, and $\leftarrow$ for right-to-left sampling from a distribution. In general, single-letter variables are set in *italic* font, function names are set in sans-serif font, and string literals are set in `slab-serif` font. We use $\mathbb{X}$ for an unspecified domain, $\mathbb{G}$ for a group, $\mathbb{Z}$ for the integers, and $\mathbb{N}$ for the natural numbers. We use λ_{c} and λ_{s} to denote the computational and statistical security parameters, respectively, and κ is the number of bits required to represent an element of the order field of an elliptic curve.[3]

Vectors and arrays are given in bold and indexed by subscripts; thus $\mathbf{a}_i$ is the i^{th} element of the vector $\mathbf{a}$, which is distinct from the scalar variable a. When we wish to select a row or column from a multi-dimensional array, we place a $*$ in the dimension along which we are not selecting. Thus $\mathbf{b}_{*,j}$ is the j^{th} column of matrix $\mathbf{b}$, $\mathbf{b}_{j,*}$ is the j^{th} row, and $\mathbf{b}_{*,*} = \mathbf{b}$ refers to the entire matrix. We use bracket notation to generate inclusive ranges, so $[n]$ denotes the integers from 1 to n and $[5,7] = \{5,6,7\}$. We use $|x|$ to denote the bit-length of x, and $|\mathbf{y}|$ to denote the number of elements in the vector $\mathbf{y}$. By convention, elliptic curve operations are expressed additively, and elliptic curve points are typically given capitalized variables.

We use $\mathcal{P}_i$ to indicate a party with index i; in a typical context, there will be a fixed set of n parties denoted $\mathcal{P}_1, \ldots, \mathcal{P}_n$. In contexts where only two parties are present, they are given indices A and B and referred to as Alice and Bob, respectively. Whenever a functionality or protocol requires a threshold of parties, it is denoted t.

2.2 Security and Communication Model

We consider a malicious PPT adversary who can statically corrupt a dishonest majority of parties. All of our proofs are expressed in the Universal Composition framework of Canetti [Can01]. We note that our techniques do not rely on any specific properties of the framework. We assume that all of the parties in any protocol are fully connected via authenticated channels.

[3] In the context of non-pairing-friendly curves, $\kappa = 2 \cdot \lambda_{\mathsf{c}}$, and all three security parameters are asymptotically equivalent.

2.3 The ECDSA Signature Scheme

We begin by describing the ECDSA signature scheme. All algorithms in the scheme are parameterized by $\mathcal{G} = (\mathbb{G}, G, q)$, which is the description of an elliptic curve group $\mathbb{G}$ of order q that is generated by G. Note that $\kappa = |q|$. Formally, we require a curve-sampling algorithm $\mathcal{G} \leftarrow \mathsf{GrpGen}(1^{\lambda_c})$, and if ECDSA is a secure signature scheme, then, at a minimum, the discrete logarithm assumption must hold with respect to the distribution of curves sampled by GrpGen.[4] In practice, the group description is fixed and standardized.

Algorithm 2.1. $\mathsf{ECDSAGen}(\mathcal{G})$

1. Uniformly choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2. Calculate the public key as $\mathsf{pk} := \mathsf{sk} \cdot G$.
3. Output $(\mathsf{pk}, \mathsf{sk})$.

Algorithm 2.2. $\mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$

1. Uniformly choose an instance key $r \leftarrow \mathbb{Z}_q$.
2. Calculate $R := r \cdot G$ and let r^{x} be the x-coordinate of R, modulo q.
3. Calculate
$$s := \frac{\mathsf{SHA2}(m) + \mathsf{sk} \cdot r^{\mathsf{x}}}{r}$$
4. Output $\sigma := (s, r^{\mathsf{x}})$.

Algorithm 2.3. $\mathsf{ECDSAVerify}(\mathcal{G}, \mathsf{pk} \in \mathbb{G}, m \in \{0,1\}^*, \sigma \in \mathbb{Z}_q^2)$

1. Parse σ as (s, r^{x}).
2. Calculate
$$R' := \frac{\mathsf{SHA2}(m) \cdot G + r^{\mathsf{x}} \cdot \mathsf{pk}}{s}$$
and let $r^{\mathsf{x}\prime}$ be the x-coordinate of R', modulo q.
3. Output 1 if and only if $r^{\mathsf{x}\prime} = r^{\mathsf{x}}$.

3 t-Party Three-Round Threshold ECDSA

We begin by giving the functionality that our threshold ECDSA protocol will realize. This functionality is fundamentally similar to the one given by Doerner

[4] This is necessary, but it is not known to be sufficient, and as of writing the security of ECDSA cannot be proven under any standard assumption.

et al. [DKLs19], but unlike their functionality, ours uses the ECDSA algorithms as black boxes, does not leak R early, and formally distinguishes *aborts*, which prevent further interactions with the functionality, from *failed signatures*, which do not. This distinction is important in threshold functionalities, because a single corrupt party should not, by participating in one signing, be able to prevent signatures from being created in the future by other groups of parties that exclude it.

In this work we do not make any assumptions about the $\mathsf{SHA2}$ function. If it is assumed to be collision resistant, then step 5 of the functionality can be changed to emit a failure when the messages themselves are unequal, rather than when their images under $\mathsf{SHA2}$ are unequal.

Functionality 3.1. $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: Threshold ECDSA**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = t$. If any party is corrupt, then the adversary $\mathcal{S}$ may instruct the functionality to abort during the setup phase *only*. $\mathcal{S}$ may also instruct the functionality to fail during the signing phase if any party indexed by $\mathbf{P}$ is corrupt, but in this case the functionality does *not* halt, and further signatures may be attempted.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, send $(\texttt{init-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{init}, \mathsf{sid})$ from all parties,

1. Sample the joint secret and public keys, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.
3. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly to $\mathcal{S}$.
4. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ from $\mathcal{S}$, send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{P}_i$ and store $(\texttt{pk-delievered}, \mathsf{sid}, i)$ in memory.

Signing: On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m_i)$ from any party $\mathcal{P}_i$, parse $\mathsf{sigid} =: \mathbf{P} \| \mathsf{sigid}'$ such that $|\mathbf{P}| = t$ and ignore the message if $i \notin \mathbf{P}$ or $\mathbf{P} \not\subseteq [n]$ or sigid is not fresh or if $(\texttt{pk-delievered}, \mathsf{sid}, i)$ does not exist in memory. Otherwise, send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, i, m_i)$ directly to $\mathcal{S}$.
On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m_i)$ from $\mathcal{P}_i$ for every $i \in \mathbf{P}$, sample $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m_{\mathbf{P}_1})$ and then

5. If there is any pair of signers $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $\mathsf{SHA2}(m_i) \neq \mathsf{SHA2}(m_j)$, then for every $i \in \mathbf{P}$, then send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$.

6. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.

7. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.

8. If no corrupt parties are indexed by $\mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P}$.

9. Once every signing party has received an output, ignore all future messages with this sigid value.

3.1 Building Blocks

In this section, we define a number of simple functionalities from which our protocol will be constructed. All are relatively standard, and they can be realized via standard techniques. In each case we give some notes on purpose and realization strategies and performance.

We begin with a functionality that samples Shamir sharings of keys for discrete-log cryptosystems (e.g. ECDSA, the Schnorr signature scheme, the ElGamal encryption scheme, the BBS+ signature scheme, etc). This functionality essentially abstracts the key generation portion of the threshold ECDSA protocols of Doerner et al. [DKLs18, DKLs19], and is nearly identical to the abstraction used by the threshold BBS+ protocol of Doerner et al. [DKL+23]. We refer the reader to the latter paper for the description of a protocol that perfectly UC-realizes the functionality in three rounds assuming ideal one-to-many committed zero-knowledge (i.e. in the $\mathcal{F}_{\mathsf{CP}}^{\mathcal{R}_{\mathsf{DL}}}$-hybrid model).

Functionality 3.2. $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$**: Discrete Log Keygen [DKL+23]**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The adversary $\mathcal{S}$ may corrupt up to $t-1$ parties that are indexed by $\mathbf{P}^*$, and if $|\mathbf{P}^*| \geq 1$, then the adversary $\mathcal{S}$ may instruct the functionality to abort.

Key Generation: On receiving $(\texttt{keygen}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, send $(\texttt{keygen-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{keygen}, \mathsf{sid})$ from all parties,

1. Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.

2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.

3. Receive $(\texttt{poly-points}, \mathsf{sid}, \{p(i)\}_{i \in \mathbf{P}^*})$ from $\mathcal{S}$.

4. Sample a random polynomial p of degree $t-1$ over $\mathbb{Z}_q$, consistent with the values $p(i)$ for $i \in \mathbf{P}^*$ that were sent by $\mathcal{S}$, and subject to $p(0) = \mathsf{sk}$.
5. For $i \in [n]$, compute $P(i) := p(i) \cdot G$.
6. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}, \{P(1), \ldots, P(n)\})$ directly to $\mathcal{S}$.
7. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ directly from $\mathcal{S}$, send $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i), \{P(1), \ldots, P(n)\})$ to $\mathcal{P}_i$.

Next, we introduce the standard commitment functionality, which can be realized in the random oracle model via a folklore method: the commitment is the image under the oracle of the committed value concatenated with a salt of length $2\lambda_\mathsf{c}$, and the decommitment is simply the committed value plus the salt.

Functionality 3.3. $\mathcal{F}_{\mathsf{Com}}$**: Commitment [CLOS02]**

In each instance one specific party $\mathcal{P}_\mathsf{S}$ commits, and the other party $\mathcal{P}_\mathsf{R}$ receives the commitment and committed value.

Commit: On receiving $(\texttt{commit}, \mathsf{sid}, x)$ from party $\mathcal{P}_\mathsf{S}$, parse $\mathsf{sid} =: \mathcal{P}_{\mathsf{S}'} \| \mathcal{P}_\mathsf{R} \| \mathsf{sid}'$. If sid is a fresh value and $\mathsf{S}' = \mathsf{S}$, then store $(\texttt{commitment}, \mathsf{sid}, x)$ in memory and send $(\texttt{committed}, \mathsf{sid})$ to $\mathcal{P}_\mathsf{R}$.

Decommit: On receiving $(\texttt{decommit}, \mathsf{sid})$ from $\mathcal{P}_\mathsf{S}$, if a record of the form $(\texttt{commitment}, \mathsf{sid}, x)$ exists in memory, then send $(\texttt{opening}, \mathsf{sid}, x)$ to $\mathcal{P}_\mathsf{R}$.

We use a functionality that non-interactively samples uniform secret-sharings of zero. It can be realized very simply in the $\mathcal{F}_{\mathsf{Com}}$-hybrid random oracle model: to initialize the protocol, each pair of parties commits and decommits a pair of λ_c-bit seeds to one another, then sums the pair to form a single shared seed. When a party invokes the protocol, it evaluates the random oracle on each of its shared seeds concatenated with the next index in sequence, and accumulates the outputs: it subtracts oracle outputs for the party pairs in which it is lower-indexed, and adds oracle outputs for the party pairs in which it is higher-indexed. The seeds can be reused indefinitely.

Functionality 3.4. $\mathcal{F}_{\mathsf{Zero}}(\mathbb{G}, n)$**: Zero-Sharing Sampling [DKL+23]**

This functionality is parameterized by the party count n and a group $\mathbb{G}$.

Sample: Upon receiving $(\texttt{sample}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, uniformly sample $\mathbf{x} \leftarrow \mathbb{G}^n$ conditioned on $\sum_{i \in [n]} x_i \equiv 0_\mathbb{G}$ and send $(\texttt{mask}, \mathsf{sid}, x_i)$ to $\mathcal{P}_i$. Upon receiving $(\texttt{sample}, \mathsf{sid})$ from any other $\mathcal{P}_j$ for $j \in [n] \setminus \{i\}$, send $(\texttt{mask}, \mathsf{sid}, x_j)$ to $\mathcal{P}_j$.

Finally, we use a random vector oblivious linear evaluation (VOLE) functionality. In this functionality, the first party (Bob) to invoke the functionality

receives a single random value; the second party (Alice) then supplies it with a vector of chosen values, and the functionality outputs to both of them secret shares of the product of the random value and each of the elements in the vector. While realizations are possible from a variety of techniques, as discussed in section 1, we suggest that this functionality be realized via a simple modification of the three-round OT-based multiplication protocol of Doerner et al. [DKLs19]. We note that the vectorization in their protocol is of a different kind than the vectorization we will construct, and consequently we ignore it (specifically, in their notation, we hardcode $\ell = 1$). We make two specific modifications. First, Bob's "adjustment value" (γ_{B} in their notation) is always hardcoded to be 0. This produces a protocol in which Bob's "input" is not chosen by him, but uniformly sampled when he is honest (and possibly nonuniform when he is corrupted), and reduces the round count to two while retaining the bandwidth advantage that Doerner et al.'s three-round multiplication protocol has over their earlier [DKLs18] two-round protocol. Second, we apply a one-sided version of the "forced-reuse" modification that Chen et al. [CCD+20] previously described, in order to force Bob to multiply his (random) input by a vector of Alice's inputs. Specifically, Alice performs the steps of the protocol for each input in her vector, but uses a single batch of Bob's OT instances for all of them, concatenating the corresponding OT payloads to form one batch of payloads with lengths proportionate to her input vector length. The resulting protocol has an initialization phase that is not represented in the functionality we intend it to realize. We have omitted the initialization from our functionality for clarity and simplicity: adapting our protocols to include it is straightforward.

Functionality 3.5. $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$**: Random Vector OLE**

This functionality interacts with two parties, $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$, who we refer to as Alice and Bob. It also interacts with the ideal adversary $\mathcal{S}$ directly. It is parameterized by a prime q that determines the order of the field over which multiplications are performed.

Sampling: On receiving $(\texttt{sample}, \mathsf{sid})$ from Bob such that $\mathsf{sid} = \mathcal{P}_{\mathsf{B}} \| \mathcal{P}_{\mathsf{A}} \| \mathsf{sid}'$ and sid is fresh and no record of the form $(\texttt{instance}, \mathsf{sid}, *, *)$ exists in memory, sample $b \leftarrow \mathbb{Z}_q$ if Bob is honest, or receive $(\texttt{adv-sample}, \mathsf{sid}, b)$ from $\mathcal{S}$ if he is corrupt, and then sample $\mathbf{c} \in \mathbb{Z}_q^\ell$ store $(\texttt{instance}, \mathsf{sid}, \mathbf{c})$, output $(\texttt{sample}, \mathsf{sid}, b)$ to Bob, and send $(\texttt{ready}, \mathsf{sid}, \mathbf{c})$ to Alice.

Multiplication: On receiving $(\texttt{multiply}, \mathsf{sid}, \mathbf{a})$ from Alice, where $\mathbf{a} \in \mathbb{Z}_q^\ell$, if there exists a message of the form $(\texttt{instance}, \mathsf{sid}, b, \mathbf{c})$ in memory, and if $(\texttt{complete}, \mathsf{sid})$ does not exist in memory, then compute $\mathbf{d}_i := \mathbf{a}_i \cdot b_i - \mathbf{c}_i \bmod q$ for every $i \in [\ell]$, send $(\texttt{products}, \mathsf{sid}, \mathbf{d})$ to Bob, and store $(\texttt{complete}, \mathsf{sid})$ in memory.

The protocol of Haitner et al. [HMRT22] may be used to realize a weaker version of the above functionality which does not always extract the corrupt

party's input, but may have reduced bandwidth costs relative to the DKLs-derived realization we have just described at the expense of one additional round. Haitner et al. claim threshold ECDSA protocols such as ours to be a primary motivation for their work. We do *not* give a precise functionality for their protocol, and we stress that we have *not* proven the combination to be secure. Nevertheless, we give a cost comparison for their protocol in section 5.2.

3.2 The Basic Three-Round Protocol

In this section we give our three round protocol. We begin by developing some intuition. Suppose that each party $\mathcal{P}_i$ knows additive shares r_i and sk_i of r and sk respectively, and samples a uniform mask ϕ_i. Suppose also that they know u_i and v_i such that

$$\sum_{i\in\mathbf{P}} u_i = \sum_{i\in\mathbf{P}} r_i \cdot \sum_{i\in\mathbf{P}} \phi_i \qquad \text{and} \qquad \sum_{i\in\mathbf{P}} v_i = \sum_{i\in\mathbf{P}} \mathsf{sk}_i \cdot \sum_{i\in\mathbf{P}} \phi_i$$

It is easy to see that given these correlations,

$$\frac{\sum_{i\in\mathbf{P}}(\mathsf{SHA2}(m)\cdot\phi_i + r^{\mathsf{x}}\cdot v_i)}{\sum_{i\in\mathbf{P}} u_i} = \frac{\mathsf{SHA2}(m) + r^{\mathsf{x}}\cdot\mathsf{sk}}{r}$$

is a valid signature on m under $\mathsf{pk} = \mathsf{sk}\cdot G$ when combined with the nonce $R = r\cdot G$. Assuming the correlation to be generated with security against malicious adversaries, it remains only to ensure that $\mathsf{pk} = \mathsf{sk}\cdot G$ and that $R = r\cdot G$, and to ensure that the adversary does not add any offsets to the correlation when the signature is assembled. The latter problem is quite simple: once m, R, and pk are fixed, there exists only one valid ECDSA signature, and so output offsets can be detected perfectly by verifying the signature after it is assembled. This leaves the problem of consistency.

Fortunately, the ingredients we need in order to ensure consistency are already present. Suppose we enrich the correlation: each party $\mathcal{P}_i$ knows $\mathbf{c}^{\mathsf{u}}_{i,j}$ and $\mathbf{c}^{\mathsf{v}}_{i,j}$ and each party $\mathcal{P}_j$ knows $\mathbf{d}^{\mathsf{u}}_{j,i}$ and $\mathbf{d}^{\mathsf{v}}_{j,i}$ such that

$$\mathbf{c}^{\mathsf{u}}_{i,j} = r_i\cdot\phi_j - \mathbf{d}^{\mathsf{u}}_{j,i} \qquad \text{and} \qquad \mathbf{c}^{\mathsf{v}}_{i,j} = \mathsf{sk}_i\cdot\phi_j - \mathbf{d}^{\mathsf{v}}_{j,i}$$

Under this correlation, if $\mathcal{P}_i$ sends $R_i = r_i\cdot G$ and $\mathsf{pk}_i = \mathsf{sk}_i\cdot G$ to $\mathcal{P}_j$, then it can also send $\mathbf{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j}\cdot G$ and $\mathbf{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j}\cdot G$ to *authenticate* the former values. Because ϕ_j is uniform and information-theoretically hidden from $\mathcal{P}_i$, if $\mathcal{P}_i$ sends $R_i \neq r_i\cdot G$, then its chance of sending $\mathbf{\Gamma}^{\mathsf{u}}_{i,j}$ satisfying

$$\mathbf{\Gamma}^{\mathsf{u}}_{i,j} = R_i\cdot\phi_j - \mathbf{d}^{\mathsf{u}}_{j,i}\cdot G$$

is negligible in κ. Thus by checking the latter equality, $\mathcal{P}_j$ can ensure that $\mathcal{P}_i$ has behaved *consistently* with overwhelming probability. A similar check allows $\mathcal{P}_j$ to ensure the consistency of pk_i and sk_i via $\mathbf{c}^{\mathsf{v}}_{i,j}$ and $\mathbf{d}^{\mathsf{v}}_{j,i}$. Finally, it is easy to compute an appropriate value u_i given knowledge of r_i, ϕ_i, $\mathbf{c}^{\mathsf{u}}_{i,*}$, and $\mathbf{d}^{\mathsf{u}}_{i,*}$, and

to compute an appropriate v_i given knowledge of sk_i, ϕ_i, $\mathbf{c}^{\mathsf{v}}_{i,*}$, and $\mathbf{d}^{\mathsf{v}}_{i,*}$, which implies that signature assembly can happen as before.

We make a few adjustments to the above basic scheme in our true protocol. First, we do not insist that each $\mathcal{P}_i$ use a consistent inversion mask ϕ_i with all of the other parties: instead, it uses an individual random mask with each counterparty and checks consistency relative to that mask, and then *adjusts* the correlation before signature assembly. This allows the correlation to be generated by a standard pairwise VOLE functionality. Second, R_i is not sent, but committed and then released, to avoid adversarial bias. Third, the shares of pk are rerandomized during each signature, in order to prevent the adversary from inducing offsets that depend on the honest parties' secrets by using its mask values inconsistently among the honest parties.

Our final protocol is three rounds. In the first round, each party commits to R_i and instantiates an $\mathcal{F}_{\mathsf{RVOLE}}$ instance toward each of the other parties. In the second round, each party decommits R_i, inputs sk_i and r_i into the instances of $\mathcal{F}_{\mathsf{RVOLE}}$ that the other parties have instantiated toward it, and sends each of the parties the values necessary to authenticate its inputs to $\mathcal{F}_{\mathsf{RVOLE}}$ and adjust the outputs of $\mathcal{F}_{\mathsf{RVOLE}}$ so that they can be assembled into a signature. After the second round, the inputs to $\mathcal{F}_{\mathsf{RVOLE}}$ are authenticated. In the third round, shares of the signature are swapped.

Protocol 3.6. $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: t-Party Three-Round ECDSA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, and the signing phase may be run many times between (varying) subsets of parties of size t. The parties in this protocol interact with the ideal functionalities $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$, $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$, and $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$. The $\mathsf{SHA2}$ function is not assumed to have any cryptographic properties.

Setup:

1. On receiving $(\texttt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks whether there exists a record of the form $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i), \{P(1), \ldots, P(n)\})$ in memory. If not, then $\mathcal{P}_i$ sends $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

2. On receiving $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i), \{P(1), \ldots, P(n)\})$ from $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$ each party $\mathcal{P}_i$ stores this message in memory and outputs $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to the environment. If $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$ aborts, then $\mathcal{P}_i$ aborts to the environment.

3. The parties perform any initialization procedure associated with $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ and $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$.[a]

Signing:

4. On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ from the environment $\mathcal{Z}$, $\mathcal{P}_i$ parses $\mathbf{P}\|\mathsf{sigid}' := \mathsf{sigid}$ such that $|\mathbf{P}| = t$, and ignores the environment's message if $i \notin \mathbf{P}$ or $\mathbf{P} \not\subseteq [n]$ or sigid is not fresh or $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i), \{P(1), \ldots, P(n)\})$ does not exist in memory. Otherwise, $\mathcal{P}_i$ continues to the next step.

5. $\mathcal{P}_i$ samples a secret instance key $r_i \leftarrow \mathbb{Z}_q$ and an inversion mask $\phi_i \leftarrow \mathbb{Z}_q$ and computes

$$\begin{aligned} R_i &:= r_i \cdot G \\ \mathbf{P}^{\text{-}j} &:= \mathbf{P} \setminus \{j\} \qquad \text{for } j \in \mathbf{P} \end{aligned}$$

6. $\mathcal{P}_i$ sends

   - $(\texttt{commit}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathsf{sigid}, R_i)$ to $\mathcal{F}_{\mathsf{Com}}$ for every $j \in \mathbf{P}^{\text{-}i}$
   - $(\texttt{sample}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathsf{sigid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for every $j \in \mathbf{P}^{\text{-}i}$
   - $(\texttt{sample}, \mathcal{P}_{\mathbf{P}_1}\| \ldots \|\mathcal{P}_{\mathbf{P}_t}\|\mathsf{sid}\|\mathsf{sigid})$ to $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$

   This completes the first round.

if pipelining, supply m here[b]

7. On receiving

   - $(\texttt{committed}, \mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid}\|\mathsf{sigid})$ from $\mathcal{F}_{\mathsf{Com}}$ for $j \in \mathbf{P}^{\text{-}i}$
   - $(\texttt{ready}, \mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid}\|\mathsf{sigid}, \{\mathbf{c}^{\mathsf{u}}_{i,j}, \mathbf{c}^{\mathsf{v}}_{i,j}\})$ from $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for $j \in \mathbf{P}^{\text{-}i}$
   - $(\texttt{sample}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathsf{sigid}, \boldsymbol{\chi}_{i,j})$ from $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for $j \in \mathbf{P}^{\text{-}i}$
   - $(\texttt{mask}, \mathcal{P}_{\mathbf{P}_1}\| \ldots \|\mathcal{P}_{\mathbf{P}_t}\|\mathsf{sid}\|\mathsf{sigid}, \zeta_i)$ from $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$

   $\mathcal{P}_i$ computes

$$\begin{aligned} \boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} &:= \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G \\ \boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} &:= \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G \\ \mathsf{sk}_i &:= p(i) \cdot \mathsf{lagrange}(\mathbf{P}, i, 0) + \zeta_i \\ \mathsf{pk}_i &:= \mathsf{sk}_i \cdot G \\ \boldsymbol{\psi}_{i,j} &:= \phi_i - \boldsymbol{\chi}_{i,j} \end{aligned}$$

   for every $j \in \mathbf{P}^{\text{-}i}$ and then $\mathcal{P}_i$ sends

   - $(\texttt{decommit}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathsf{sigid})$ to $\mathcal{F}_{\mathsf{Com}}$
   - $(\texttt{multiply}, \mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid}\|\mathsf{sigid}, \{r_i, \mathsf{sk}_i\})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$
   - $(\texttt{check-adjust}, \mathsf{sid}, \mathsf{sigid}, \boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}, \boldsymbol{\Gamma}^{\mathsf{v}}_{i,j}, \boldsymbol{\psi}_{i,j}, \mathsf{pk}_i)$ to $\mathcal{P}_j$

for every $j \in \mathbf{P}^{-i}$.

if preprocessing, supply m here[b]

8. On receiving

   - $(\texttt{opening}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, R_j)$ from $\mathcal{F}_{\mathsf{Com}}$
   - $(\texttt{products}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{d}^{\mathsf{u}}_{i,j}, \mathbf{d}^{\mathsf{v}}_{i,j}\})$ from $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$
   - $(\texttt{check-adjust}, \mathsf{sid}, \mathsf{sigid}, \boldsymbol{\Gamma}^{\mathsf{u}}_{j,i}, \boldsymbol{\Gamma}^{\mathsf{v}}_{j,i}, \boldsymbol{\psi}_{j,i}, \mathsf{pk}_j)$ from $\mathcal{P}_j$

   for every $j \in \mathbf{P}^{-i}$, $\mathcal{P}_i$ checks whether

$$\begin{aligned} \boldsymbol{\chi}_{i,j} \cdot R_j - \boldsymbol{\Gamma}^{\mathsf{u}}_{j,i} &= \mathbf{d}^{\mathsf{u}}_{i,j} \cdot G \\ \boldsymbol{\chi}_{i,j} \cdot \mathsf{lagrange}(\mathbf{P}, j, 0) \cdot P(j) - \boldsymbol{\Gamma}^{\mathsf{v}}_{j,i} &= \mathbf{d}^{\mathsf{v}}_{i,j} \cdot G \end{aligned}$$

   for every $j \in \mathbf{P}^{-i}$, and whether

$$\sum_{k \in \mathbf{P}} \mathsf{pk}_k = \mathsf{pk}$$

   and if these equations hold, then $\mathcal{P}_i$ computes

$$\begin{aligned} R &\coloneqq \sum_{j \in \mathbf{P}} R_j \\ u_i &\coloneqq r_i \cdot \left(\phi_i + \sum_{j \in \mathbf{P}^{-i}} \boldsymbol{\psi}_{j,i}\right) + \sum_{j \in \mathbf{P}^{-i}} (\mathbf{c}^{\mathsf{u}}_{i,j} + \mathbf{d}^{\mathsf{u}}_{i,j}) \\ v_i &\coloneqq \mathsf{sk}_i \cdot \left(\phi_i + \sum_{j \in \mathbf{P}^{-i}} \boldsymbol{\psi}_{j,i}\right) + \sum_{j \in \mathbf{P}^{-i}} (\mathbf{c}^{\mathsf{v}}_{i,j} + \mathbf{d}^{\mathsf{v}}_{i,j}) \\ w_i &\coloneqq \mathsf{SHA2}(m) \cdot \phi_i + r^{\mathsf{x}} \cdot v_i \end{aligned}$$

   where r^{x} is the x-coordinate of R, and sends $(\texttt{fragment}, \mathsf{sid}, \mathsf{sigid}, w_i, u_i)$ to $\mathcal{P}_j$ for every $j \in \mathbf{P}^{-i}$. On the other hand, if $\mathcal{P}_i$'s shared instance of $\mathcal{F}_{\mathsf{RVOLE}}$ with $\mathcal{P}_j$ aborts, or if any of the aforementioned equations do not hold for some $j \in \mathbf{P}^{-i}$, then $\mathcal{P}_i$ sends $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ to all other parties *and sends an analogous message at the corresponding point in all concurrent signing sessions that involve* $\mathcal{P}_j$, outputs $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to the environment, does not continue to step 10, and does not participate in any future signature signing sessions involving $\mathcal{P}_j$. This completes the third round.

9. On receiving $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ from any $\mathcal{P}_j$ for $j \in \mathbf{P}$, $\mathcal{P}_i$ outputs $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to the environment, and does not continue to step 10.

10. On receiving $(\texttt{fragment}, \mathsf{sid}, \mathsf{sigid}, w_j, u_j)$ from $\mathcal{P}_j$ for every $j \in \mathbf{P}^{-i}$, $\mathcal{P}_i$ computes

$$s := \frac{\sum_{j \in \mathbf{P}} w_j}{\sum_{j \in \mathbf{P}} u_j}$$

and outputs $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (s, r^{\mathsf{x}}))$ to the environment if and only if $\mathsf{ECDSAVerify}(\mathcal{G}, \mathsf{pk}, m, (s, r^{\mathsf{x}})) = 1$; otherwise, $\mathcal{P}_i$ outputs $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to the environment.

[a] The functionalities have no such initialization per se, but their realizations might, and this is the appropriate time to do it.

[b] In this case, the signing phase is initiated with a $(\texttt{pre-sign}, \mathsf{sid}, \mathsf{sigid})$ message from the environment, and waits at the indicated point for a $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ message from the enviornment. See section 3.3.

3.3 Pipelining and Preprocessing

We have marked the above protocol in two places to show how it can be modified to add *pipelining* or *preprocessing*. In each case, the parties must supply the message m to the protocol at the indicated point, instead of at the beginning of the protocol.

Pipelining. Pipelining allows the first round of the protocol to be evaluated before the message is known. If a single group of parties signs many messages together, they can evaluate the first round of one signing instance along simultaneously with the third round of a previous signature. This allows the signing procedure to be completed with only two rounds of latency. Because the nonce R is not defined until the second round, the standard order of quantifiers, in which the message cannot depend upon R is respected, and the output signatures are secure if single-party ECDSA signatures are. However, R becomes well-defined from the point of view of the adversary as soon as the the honest parties are activated by the environment, and potentially before the corrupt parties are. This necessitates a revised functionality, which we present below.

Functionality 3.7. $\mathcal{F}_{\mathsf{PipelinableECDSA}}(\mathcal{G}, n, t)$**: Pipelineable TECDSA**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = t$. If any party is corrupt, then the adversary $\mathcal{S}$ may instruct the functionality to abort during the setup phase. $\mathcal{S}$ may also instruct the functionality to fail during the signing phase if any party indexed by $\mathbf{P}$ is corrupt, but in this case the functionality does *not* halt, and further signatures may be attempted.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, send $(\texttt{init-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{init}, \mathsf{sid})$ from all parties,

1. Sample the joint secret and public keys, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.
3. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly to $\mathcal{S}$.
4. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ from $\mathcal{S}$, send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{P}_i$ and store $(\texttt{pk-delievered}, \mathsf{sid}, i)$ in memory.

Signing: On receiving $(\texttt{pre-sign}, \mathsf{sid}, \mathsf{sigid})$ from any party $\mathcal{P}_i$, parse $\mathsf{sigid} =: \mathbf{P} \| \mathsf{sigid}'$ such that $|\mathbf{P}| = t$ and ignore the message if $i \notin \mathbf{P}$ or $\mathbf{P} \not\subseteq [n]$ or sigid is not fresh or if $(\texttt{pk-delievered}, \mathsf{sid}, i)$ does not exist in memory. Otherwise, send $(\texttt{presig-req}, \mathsf{sid}, \mathsf{sigid}, i)$ directly to $\mathcal{S}$ and store $(\texttt{ready}, \mathsf{sid}, \mathsf{sigid}, i)$ in memory.
On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ from $\mathcal{P}_i$ for some $i \in \mathbf{P}$, if $(\texttt{ready}, \mathsf{sid}, \mathsf{sigid}, j)$ exists in memory for all $j \in \mathbf{P}$ then

5. If $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ does not exist in memory, then sample $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$, store $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ in memory, and if at least one party indexed by $\mathbf{P}$ is corrupt, then send $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, r^{\times})$ directly to $\mathcal{S}$.
6. If at least one party indexed by $\mathbf{P}$ is corrupt, then send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, i, m)$ directly to $\mathcal{S}$.

Once every $\mathcal{P}_i$ for $i \in \mathbf{P}$ has sent $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$,

7. If the value of m submitted is not consistent among all parties, then for every $i \in \mathbf{P}$, wait for $\mathcal{S}$ $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ and then send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$.
8. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.
9. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.
10. If no corrupt parties are indexed by $\mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P}$.

11. Once every signing party has received an output, ignore all future messages with this sigid value.

Preprocessing. Preprocessing allows the first *two* rounds of the protocol to be evaluated before the message is known, which leaves only the last round (containing nothing but a few simple field operations and one signature verification per party) as the only round that must be evaluated online. Unlike pipelining, preprocessing does not preserve the standard order of quantifiers: the environment can potentially condition the message on R, which is fixed in the second round. Groth and Shoup [GS22b] gave a proof under a new assumption on SHA2 in a variant of the generic group model that ECDSA is secure even if this occurs. They also show a number of conditions under which preprocessing can lead to attacks (none of which apply to our protocol, as presented). We warn that preprocessing should only be used in practice by those who understand and accept the implications and risks associated with it. Nevertheless, our protocol is compatible with it.

3.4 Comparison to DKLs19

The protocol presented in this section is a significant revision of the one presented by Doerner et al. [DKLs19]; although it contain many of the same fundamental ideas, a rearrangement of the main protocol structure and the elimination of an intermediate functionality (the so called inverse-sampling functionality) have yielded a significant improvement in the number of rounds and a completely new proof under strictly weaker assumptions. Specifically, whereas the 2019 protocol required either $6 + \log t$ or 10 rounds under the computational Diffie-Hellman assumption in the non-programmable global random oracle model, our new protocol requires only 3 rounds (one of which is pipelineable) and is statistically secure in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{DLKeyGen}})$-hybrid model. Our new protocol has bandwidth requirements similar to the 10-round version of the 2019 protocol, whereas the $(6 + \log t)$-round version required slightly less bandwidth.

The heart of the structural change lies in the way the two protocols compute shares of $1/r$ and sk/r, and in the way they check the correctness of these computations. In the 2019 protocol, a distinct functionality was defined to sample R along with shares of r and $1/r$. This functionality was realized by a protocol that sampled multiplicative shares of r, inverted them locally, and then used a $O(\log t)$-long sequence of pairwise $\mathcal{F}_{\mathsf{Mul}}$ invocations to compute additive shares of both r and ϕ/r, where ϕ is a uniform mask. A single commit-and-release check assures the well-formedness of the shares in the 2019 scheme, and then the shares are unmasked. Only once shares of $1/r$ are known are they multiplied by shares of sk, with an additional commit-and-releaes check (containing two check values) establishing the correctness of this multiplication with respect to pk. The 2019 protocol's higher round count is due the fact that it performs the inversion and multiplication operations sequentially, and the fact

that it performs two sequential commit-and-release checks.

In contrast the protocol introduced here performs inversion, multiplication with sk, and consistency checking simultaneously. The consistency properties of the $\mathcal{F}_{\mathsf{RVOLE}}$ functionality are used to guarantee that each party $\mathcal{P}_j$ multiplies a single uniform value $\boldsymbol{\chi}_{j,i}$ by both r_i and sk_i for every $\mathcal{P}_i$ such that $i \neq j$. This value $\boldsymbol{\chi}_{j,i}$ first serves as a MAC key for the pair of parties in question: they check the MAC in the exponent to ensure that $\mathcal{P}_i$'s inputs to $\mathcal{F}_{\mathsf{RVOLE}}$ are consistent with R_i and pk_i, respectively. This check is statistical. Simultaneously, $\mathcal{P}_j$ publishes the offset $\boldsymbol{\psi}_{j,i}$ between every $\boldsymbol{\chi}_{j,i}$ and a single uniform inversion nonce ϕ_i. If all parties do this honestly, then their shares can be converted into the form requires by the inversion trick of Bar-Ilan and Beaver, and assembly of the signature can be done just as it was in prior works that use a similar correlation [ANO+22, GS22a]. If any parties cheat in this adjustment, the induced offsets can be perfectly simulated independently of the honest parties' secrets. Because fixing m, R, and pk fixes the value of the signature, verifying the signature becomes a perfect check for malicious behavior in the final adjustment and assembly stage of the protocol.

While the number of rounds is significantly improved relative to the 2019 scheme, we note that the number of elliptic curve scalar operations grows with the number of signers in our new scheme, whereas in the 2019 scheme it was a constant. We believe that in most deployment scenarios this trade is beneficial. In addition to the above changes, key generation process has been abstracted via a new ideal functionality.

3.5 Two-Party Two-Message ECDSA

In addition to their general t-of-n protocol, Doerner et al. also proposed a specialized 2-of-n protocol [DKLs18] that required only one message to be sent in each direction. When $t = 2$, a simple modification of our new protocol allows it to match the communication properties of theirs. In each signing instance, one of the two parties is chosen as the initiator. We will label the initiator as Alice, and the other party as Bob. Only Alice will receive the signature at the end. The parties run π_{ECDSA} with pipelining, as described in sections 3.2 and 3.3, and make the following modifications:

1. Alice's pipelined first message is not triggered by any message from the environment. Instead, she sends her first message with her second message, upon receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid})$ from the environment.

2. Bob's second message is not triggered by a $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid})$ message from the environment. Instead, upon receiving Alice's first and second messages, Bob outputs $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid})$ to the environment, and sends his second message only after the environment responds with $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, m)$.

3. Bob sends his third message at the same time he sends his second message. Since Alice's second message has already been received, this is possible.

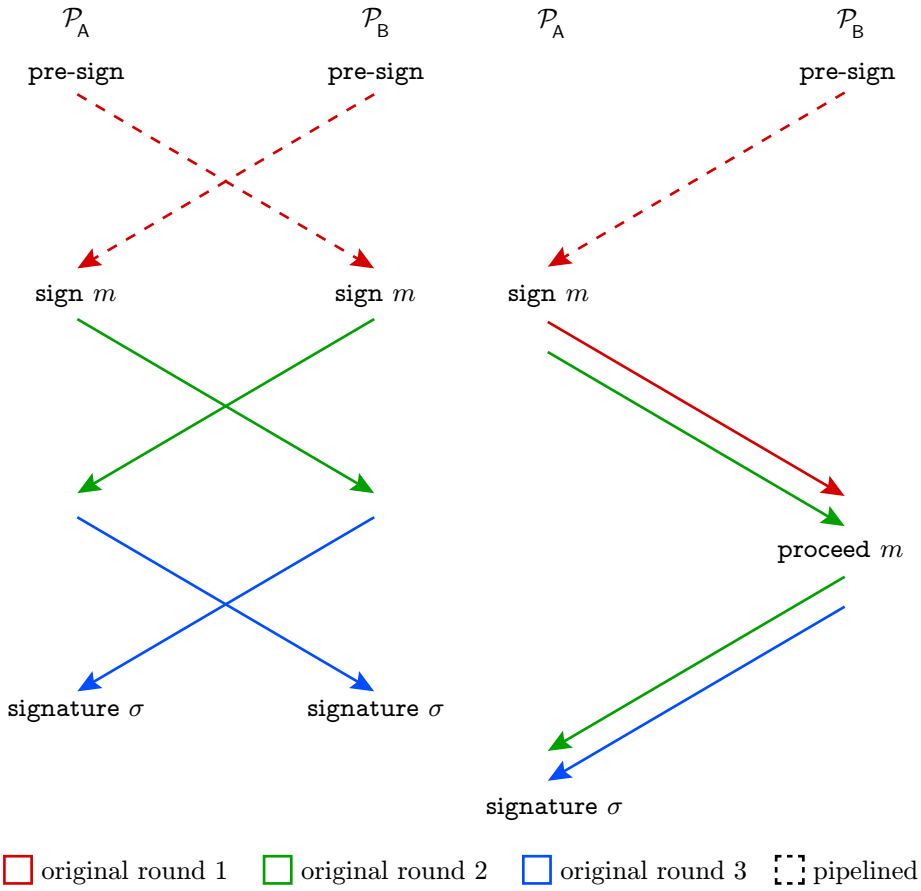

Figure 1: Two-party Message Structures Illustrated. On the left is the protocol structure, with pipelining, as described in sections 3.2 and 3.3. On the right is the protocol structure suggested for the two-party setting in this section.

4. Alice never sends her third message, depriving Bob of the s component of the output signature.

We note that these modifications to the protocol are secure because they are essentially equivalent to rushing behavior, and our proof in section 4 already acounts for rushing adversaries. We illustrate the modifications in figure 1. The resulting protocol comprises three messages, and if the parties pipeline the messages of each signature to occur simultaneously with the last message of a previous signature, then the resulting protocol has two messages in effect, just like the original 2-of-n protocol.

Comparison to DKLs18. Compared to the original 2-of-n DKLs protocol from 2018, our new protocol requires pipelining (and thus the storage of intermediate state) in order to achieve a two-round structure. We note that in the two party-case the downside implied by this is minimal: the stored state is exclusively pairwise, just like the stored state already required by the OT-extension protocol that is used to realize $\mathcal{F}_{\mathsf{RVOLE}}$. On the other hand, our new protocol realizes a *standard* threshold signing functionality, whereas the 2018 DKLs protocol realizes a weaker functionality that allows the adversary to bias R, and that is only known to be equivalent to the standard functionality in the generic group model. Moreover, our protocol is statistically secure, whereas the 2018 protocol required a reduction to the computational Diffie-Hellman assumption on $\mathbb{G}$, and a reduction to the forgery game for ECDSA. Finally, our protocol improves upon the efficiency of the 2018 protocol. We do not make use of zero-knowledge during signing, whereas the 2018 protocol does. In the UC paradigm, realizing a zero-knowledge functionality in one round (as required by the 2018 paper) requires a straight-line extractor such as the Fischlin [Fis05] or Kondi-shelat [Ks22] transform. This is by far the most computationally-expensive component of the 2018 protocol, and the only reason that the 2018 protocol requires a superconstant number of public-key operations during signing. We eliminate this cost. We also improve upon the bandwidth of the 2018 protocol: the chosen-input multiplication subprotocol used in that work requires a total of $4\kappa+4\lambda_{\mathsf{s}}$ OT instances, half of which have a payload size of 2κ and half of which have a payload size of 4κ. Realizing the randomized VOLE instances required by our new protocol via the DKLs-derived VOLE protocol suggested in section 3.1 requires a total of $2\kappa + 4\lambda_{\mathsf{s}}$ OT instances, all of which have a payload size of 4κ. When $\kappa = 256$ and $\lambda_{\mathsf{s}} = 80$, as is common in practice, this yields a 17.5% savings in the bandwidth due to the OT payload. This improvement is independent of improvements due to new OT-extension techniques, and independent of the random-oracle based optimization that can be applied to both their multiplication protocol and our VOLE protocol, which is discussed in section 5.1.

4 Proof of Security for t-Party ECDSA

In section 1, we stated the following security theorem for our protocol:

Theorem 1.1 (Informal Security Theorem)**.** *In the* $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{Zero}}, \mathcal{F}_{\mathsf{RVOLE}}, \mathcal{F}_{\mathsf{DLKeyGen}})$*-hybrid model,* $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *statistically UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *against a malicious adversary that statically corrupts up to* $t-1$ *parties.*

There is, however, one caveat we must address when formalizing the above theorem. The UC model officially captures only a computational notion of security, and if it is extended to permit unbounded environments and adversaries, then a problem arises when considering protocols that realize reactive functionalities such as $\mathcal{F}_{\mathsf{ECDSA}}$: if each invocation implies a statistically negligible chance of distinguishing the real and ideal worlds, but the environment

is allowed exponentially-many invokations, then, the overall probability of distinguishing such a protocol from its functionality becomes noticeable. To avoid this, we enforce explicit (but arbitrary) polynomial bounds on both the number of parties and the number of times the honest parties may be invoked, while allowing the environment to be otherwise unbounded. Thus we have the following formal theorem:

Theorem 4.1 (Formal Security Theorem)**.** *For every malicious adversary $\mathcal{A}$ that statically corrupts up to $t-1$ parties, there exists a PPT simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$ and every pair of polynomials μ, ν, if $\mu(\lambda)$ bounds the number of times $\mathcal{Z}$ invokes any honest party, then*

$$\left\{\textsc{Real}_{\pi_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{A},\mathcal{Z}}\left(\lambda, z\right) : \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda)\right\}_{\substack{\lambda\in\mathbb{N}, n\in[2,\nu(\lambda)],\\ t\in[2,n], z\in\{0,1\}^*}}$$
$$\approx_{\mathrm{s}} \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{Z}}\left(\lambda, z\right) : \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda)\right\}_{\substack{\lambda\in\mathbb{N}, n\in[2,\nu(\lambda)],\\ t\in[2,n], z\in\{0,1\}^*}}$$

Proof. We begin by specifying the simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$, after which we will give a sequence of hybrid experiments to establish that it produces a view for the environment that is indistinguishable from the real world.

Simulator 4.2. $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: t-Party ECDSA**

This simulator is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The simulator has oracle access to the adversary $\mathcal{A}$, and emulates for it an instance of the protocol $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ involving the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. When the emulated protocol instance begins, $\mathcal{A}$ announces the identities of up to $t-1$ corrupt parties. Let the indices of these parties be given by $\mathbf{P}^* \subseteq [n]$. $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of every corrupt party, and in the exeriment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and the corrupt parties on behalf of every honest party and on behalf of the ideal oracles $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$, $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$, and $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

Setup:

1. On receiving $(\texttt{keygen}, \mathsf{sid})$ from $\mathcal{P}_i$ for some $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$, send
   - $(\texttt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_i$
   - $(\texttt{keygen-req}, \mathsf{sid}, i)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$
2. On receiving $(\texttt{init-req}, \mathsf{sid}, j)$ for some $j \in [n] \setminus \mathbf{P}^*$ directly from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$, send $(\texttt{keygen-req}, \mathsf{sid}, j)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

3. On receiving $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$, forward this message to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

4. On receiving $(\texttt{release}, \mathsf{sid}, i)$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$, forward this message directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.

5. On receiving $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_j$ for $j \in \mathbf{P}^*$, if $\mathcal{P}_j$ is the first party for which such a message was received, then

   (a) Wait to receive $(\texttt{poly-points}, \mathsf{sid}, \{p(i)\}_{i \in \mathbf{P}^*})$ directly from $\mathcal{A}$.

   (b) Choose $\mathbf{P}^{\mathsf{sk}} \subset [n]$ such that $\mathbf{P}^* \subseteq \mathbf{P}^{\mathsf{sk}}$ and $|\mathbf{P}^{\mathsf{sk}}| = t - 1$.

   (c) For every $i \in \mathbf{P}^{\mathsf{sk}} \setminus \mathbf{P}^*$ sample $p(i) \leftarrow \mathbb{Z}_q$ uniformly, and then for every $j \in \mathbf{P}^{\mathsf{sk}}$, compute $P(i) := p(i) \cdot G$.

   (d) For every $k \in [n] \setminus \mathbf{P}^{\mathsf{sk}}$, compute

   $$P(k) := \frac{\mathsf{pk} - \sum_{j \in \mathbf{P}^{\mathsf{sk}}} \mathsf{lagrange}(\mathbf{P}^{\mathsf{sk}} \cup \{k\}, j, 0) \cdot P(j)}{\mathsf{lagrange}(\mathbf{P}^{\mathsf{sk}} \cup \{k\}, k, 0)}$$

   and store $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}, \{P(1), \ldots, P(n)\})$ in memory.

   (e) Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}, \{P(1), \ldots, P(n)\})$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

   and regardless, send $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i), \{P(1), \ldots, P(n)\})$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$.

6. Initialize the blacklist for sid to be empty.

Signing:

7. On receiving $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, j, m_j)$ from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of the corrupt signers, compute

$$\begin{aligned}
\mathbf{P} \| \mathsf{sigid}' &:= \mathsf{sigid} \quad \text{such that } |\mathbf{P}| = t \\
\mathbf{C} &:= \mathbf{P} \cap \mathbf{P}^* \\
\mathbf{H} &:= \mathbf{P} \setminus \mathbf{C} \\
\mathbf{P}^{-k} &:= \mathbf{P} \setminus \{k\} \qquad \text{for } k \in \mathbf{P} \\
\mathbf{c}^{\mathsf{u}}_{i,j} &\leftarrow \mathbb{Z}_q \\
\mathbf{c}^{\mathsf{v}}_{i,j} &\leftarrow \mathbb{Z}_q
\end{aligned}$$

   and if there is any $i \in \mathbf{P}^{-j}$ such that (j, i) is in the blacklist for sid, then ignore these messages and act as though they had never arrived. Otherwise, for every $i \in \mathbf{C}$ send to $\mathcal{P}_i$

- $(\texttt{committed}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}$
- $(\texttt{ready}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{c}^{\mathsf{u}}_{i,j}, \mathbf{c}^{\mathsf{v}}_{i,j}\})$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$

8. Upon receiving $(\texttt{sample}, \mathcal{P}_{\mathbf{P}_1} \| \ldots \| \mathcal{P}_{\mathbf{P}_t} \| \mathsf{sid} \| \mathsf{sigid})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$, sample $\zeta_i \leftarrow \mathbb{Z}_q$ and respond with $(\texttt{mask}, \mathcal{P}_{\mathbf{P}_1} \| \ldots \| \mathcal{P}_{\mathbf{P}_t} \| \mathsf{sid} \| \mathsf{sigid}, \zeta_i)$ on behalf of $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$. Note that this step may occur at any time.

9. Upon satisfying satisfying step 7 for every $j \in \mathbf{H}$ and also receiving

   - $(\texttt{commit}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid}, \mathbf{R}_{i,j})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com}}$
   - $(\texttt{sample}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$
   - $(\texttt{adv-sample}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid}, \boldsymbol{\chi}_{i,j})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$

   for every $i \in \mathbf{C}$ and some consistent $j \in \mathbf{H}$, if sigid is fresh and the records $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}, \{P(i)\}_{i \in [n]})$ and $(\texttt{secret-key}, \mathsf{sid}, i, p(i))$ for every $i \in \mathbf{C}$ are stored in memory, then

   - if $\mathcal{P}_j$ is *not* the last honest party for whom these conditions hold, then sample $r_j \leftarrow \mathbb{Z}_q$, $\mathsf{sk}_j \leftarrow \mathbb{Z}_q$, $\phi_j \leftarrow \mathbb{Z}_q$, $\delta^{\mathsf{u}}_j \leftarrow \mathbb{Z}_q$, and $\delta^{\mathsf{v}}_j \leftarrow \mathbb{Z}_q$, and compute $R_j := r_j \cdot G$ and $\mathsf{pk}_j := \mathsf{sk}_j \cdot G$
   - if $\mathcal{P}_j$ *is* the last honest party for whom these conditions hold, then let $h := j$. For every $i \in \mathbf{C}$, send $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m_h)$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_i$ and send $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$. If $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, (s, r^{\mathsf{x}}))$ is received in reply on behalf of the corrupt parties, reconstruct R from the x-coordinate r^{x} and compute

   $$R_h := R - \sum_{k \in \mathbf{P}^{-h}} R_k \qquad \text{and} \qquad \mathsf{pk}_h := \mathsf{pk} - \sum_{k \in \mathbf{P}^{-h}} \mathsf{sk}_k \cdot G$$

   If $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ is received in reply, then sample $R \leftarrow \mathbb{G}$ and $s \leftarrow \mathbb{Z}_q$ uniformly and compute R_h and pk_h as above.

   and then for every $i \in \mathbf{C}$ compute

   $$\begin{aligned}
   \boldsymbol{\psi}_{j,i} &\leftarrow \mathbb{Z}_q \\
   \mathbf{d}^{\mathsf{u}}_{i,j} &\leftarrow \mathbb{Z}_q \\
   \mathbf{d}^{\mathsf{v}}_{i,j} &\leftarrow \mathbb{Z}_q \\
   \boldsymbol{\Gamma}^{\mathsf{u}}_{j,i} &:= \boldsymbol{\chi}_{i,j} \cdot R_j - \mathbf{d}^{\mathsf{u}}_{i,j} \cdot G \\
   \boldsymbol{\Gamma}^{\mathsf{v}}_{j,i} &:= \boldsymbol{\chi}_{i,j} \cdot \mathsf{pk}_j - \mathbf{d}^{\mathsf{v}}_{i,j} \cdot G
   \end{aligned}$$

   and send to $\mathcal{P}_i$

   - $(\texttt{products}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{d}^{\mathsf{u}}_{i,j}, \mathbf{d}^{\mathsf{v}}_{i,j}\})$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$

- $(\texttt{opening}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, R_j)$ on behalf of $\mathcal{F}_{\mathsf{Com}}$
- $(\texttt{check-adjust}, \mathsf{sid}, \mathsf{sigid}, \boldsymbol{\Gamma}^{\mathsf{u}}_{j,}, \boldsymbol{\Gamma}^{\mathsf{v}}_{j,i}, \boldsymbol{\psi}_{j,i}, \mathsf{pk}_j)$ on behalf of $\mathcal{P}_j$

10. Upon satisfying satisfying steps 7 and 9 for every $j \in \mathbf{H}$ and receiving $(\texttt{abort}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid})$ directly from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for some $i \in \mathbf{C}$ and some $j \in \mathbf{H}$, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ to all corrupt parties on behalf of $\mathcal{P}_j$, *send the* `fail` *message on behalf of* $\mathcal{P}_j$ *at the corresponding point in all concurrent signing sessions involving* $\mathcal{P}_j$ *and* $\mathcal{P}_i$, append (j, i) to the blacklist for sid, and ignore all future instructions pertaining to the signature ID sigid.

11. Upon satisfying satisfying steps 7 and 9 for every $j \in \mathbf{H}$ and receiving

    - $(\texttt{decommit}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}$
    - $(\texttt{multiply}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{a}^{\mathsf{u}}_{i,j}, \mathbf{a}^{\mathsf{v}}_{i,j}\})$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$
    - $(\texttt{check-adjust}, \mathsf{sid}, \mathsf{sigid}, \boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}, \boldsymbol{\Gamma}^{\mathsf{v}}_{i,j}, \boldsymbol{\psi}_{i,j}, \mathbf{pk}_{i,j})$ on behalf of $\mathcal{P}_j$

    for every $i \in \mathbf{C}$ and some consistent $j \in \mathbf{H}$,

    - If there exists some $i \in \mathbf{C}$ such that $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G \neq \mathbf{R}_{i,j}$ or $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G \neq \mathbf{pk}_{i,j}$ or $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} \neq \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ or $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} \neq \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$, or if

      $$\sum_{i \in \mathbf{C}} \mathbf{pk}_{i,j} + \sum_{k \in \mathbf{H}} \mathsf{pk}_k \neq \mathsf{pk}$$

      then send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, k)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ for every $k \in \mathbf{H}$, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ to all corrupt parties on behalf of $\mathcal{P}_j$, *send the* `fail` *message on behalf of* $\mathcal{P}_j$ *at the corresponding point in all concurrent signing sessions involving* $\mathcal{P}_j$ *and* $\mathcal{P}_i$, append (j, i) to the blacklist for sid, and ignore all future instructions pertaining to the signature ID sigid.
    - If $j \neq h$ and $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G = \mathbf{R}_{i,j}$ and $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G = \mathbf{pk}_{i,j}$ and $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$ for every $i \in \mathbf{C}$, and if

      $$\sum_{i \in \mathbf{C}} \mathbf{pk}_{i,j} + \sum_{k \in \mathbf{H}} \mathsf{pk}_k = \mathsf{pk}$$

      then compute

      $$\begin{aligned} u_j &:= r_j \cdot \sum_{i \in \mathbf{C}} \boldsymbol{\psi}_{i,j} + \delta^{\mathsf{u}}_j \\ &\quad + \sum_{i \in \mathbf{C}} \left((\boldsymbol{\phi}_j - \boldsymbol{\psi}_{j,i}) \cdot \mathbf{a}^{\mathsf{u}}_{i,j} + \boldsymbol{\chi}_{i,j} \cdot r_j - \mathbf{c}^{\mathsf{u}}_{i,j} - \mathbf{d}^{\mathsf{u}}_{i,j} \right) \\ v_j &:= \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \boldsymbol{\psi}_{i,j} + \delta^{\mathsf{v}}_j \end{aligned}$$

$$+ \sum_{i \in \mathbf{C}} \left((\boldsymbol{\phi}_j - \boldsymbol{\psi}_{j,i}) \cdot \mathbf{a}^{\mathsf{v}}_{i,j} + \boldsymbol{\chi}_{i,j} \cdot \mathsf{sk}_j - \mathbf{c}^{\mathsf{v}}_{i,j} - \mathbf{d}^{\mathsf{v}}_{i,j} \right)$$

$$w_j \coloneqq \mathsf{SHA2}(m_h) \cdot \phi_j + r^{\mathsf{x}} \cdot v_j$$

and send $(\texttt{fragment}, \mathsf{sid}, \mathsf{sigid}, w_j, u_j)$ to $\mathcal{P}_i$ for every $i \in \mathbf{C}$ on behalf of $\mathcal{P}_j$.

- If $j = h$ and $\mathbf{a}^{\mathsf{u}}_{i,h} \cdot G = \mathbf{R}_{i,h}$ and $\mathbf{a}^{\mathsf{v}}_{i,h} \cdot G = \mathbf{pk}_{i,h}$ and $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,h} = \mathbf{c}^{\mathsf{u}}_{i,h} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,h} = \mathbf{c}^{\mathsf{v}}_{i,h} \cdot G$ for every $i \in \mathbf{C}$, and if

$$\sum_{i \in \mathbf{C}} \mathbf{pk}_{i,j} + \sum_{k \in \mathbf{H}} \mathsf{pk}_k = \mathsf{pk}$$

then sample $u_h \leftarrow \mathbb{Z}_q$ and compute

$$\phi_i \coloneqq \boldsymbol{\psi}_{i,h} + \boldsymbol{\chi}_{i,h} \qquad \text{for } i \in \mathbf{C}$$

$$\hat{u} \coloneqq \sum_{i \in \mathbf{C}} \left(\mathbf{a}^{\mathsf{u}}_{i,h} \cdot \Big(\sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + \boldsymbol{\psi}_{h,i} \Big) + \mathbf{c}^{\mathsf{u}}_{i,h} + \mathbf{d}^{\mathsf{u}}_{i,h} \right) + \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta^{\mathsf{u}}_j + r_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right)$$

$$\hat{v} \coloneqq \sum_{i \in \mathbf{C}} \left(\mathbf{a}^{\mathsf{v}}_{i,h} \cdot \Big(\sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + \boldsymbol{\psi}_{h,i} \Big) + \mathbf{c}^{\mathsf{v}}_{i,h} + \mathbf{d}^{\mathsf{v}}_{i,h} \right) + \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta^{\mathsf{v}}_j + \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right)$$

$$\hat{w} \coloneqq \mathsf{SHA2}(m_h) \cdot \sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + r^{\mathsf{x}} \cdot \hat{v}$$

$$w_h \coloneqq s \cdot u_h + s \cdot \hat{u} - \hat{w}$$

and send $(\texttt{fragment}, \mathsf{sid}, \mathsf{sigid}, w_h, u_h)$ to $\mathcal{P}_i$ for every $i \in \mathbf{C}$ on behalf of $\mathcal{P}_h$.

12. On receiving $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ from $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ for some $j \in \mathbf{H}$ and some $i \in \mathbf{C}$, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, j)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.

13. On receiving $(\texttt{fragment}, \mathsf{sid}, \mathsf{sigid}, \mathbf{w}_{i,j}, \mathbf{u}_{i,j})$ from $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ for some $j \in \mathbf{H}$ and *every* $i \in \mathbf{C}$, if

$$\frac{\sum_{k \in \mathbf{H}} w_k + \sum_{i \in \mathbf{C}} \mathbf{w}_{i,j}}{\sum_{k \in \mathbf{H}} u_k + \sum_{i \in \mathbf{C}} \mathbf{u}_{i,j}} = s$$

then send $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, j)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$; otherwise, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, j)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.

Our sequence of hybrid experiments begins with the real world

$$\mathcal{H}_0 = \left\{\text{REAL}_{\pi_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{A},\mathcal{Z}}\,(\lambda, z) : \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda)\right\}_{\substack{\lambda\in\mathbb{N}, n\in[2,\nu(\lambda)],\\ t\in[2,n], z\in\{0,1\}^*}}$$

and proceeds by gradually replacing the code of the real parties with elements of the simulator until $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ is be fully implemented and the experiment is be the ideal one.

Hybrid $\mathcal{H}_1$. This hybrid experiment replaces all of the individual honest parties and ideal functionalities in $\mathcal{H}_0$ with a single simulator machine $\mathcal{S}$ that runs their code and interacts with the adversary, environment, and corrupt parties on their behalf. Since $\mathcal{S}$ interacts with the adversarial entities on behalf of the ideal functionalities, it learns any values they receive or that are defined by their internal state (for example, the secret key sk). This is a purely syntactical change, and so it must be the case that $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This hybrid behaves identically to $\mathcal{H}_1$, except that the consistency checks that are performed by $\mathcal{S}$ on behalf of the *honest* parties in step 8 of π_{ECDSA} are replaced. Let $\mathbf{R}_{i,j}$ denote the value of R_i actually transmitted from party $\mathcal{P}_i$ to party $\mathcal{P}_j$ (although the protocol specifies that $\mathcal{P}_i$ should use a single consistent value R_i with all honest parties, $\mathcal{P}_i$ might use inconsistent values if it is corrupt and misbehaves). Similarly, let $\mathbf{pk}_{i,j}$ be the value of pk_i actually transmitted to $\mathcal{P}_j$. In $\mathcal{H}_2$, $\mathcal{S}$ does not check whether

$$\boldsymbol{\chi}_{j,i} \cdot \mathbf{R}_{i,j} - \boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{d}^{\mathsf{u}}_{j,i} \cdot G \tag{1}$$

$$\boldsymbol{\chi}_{j,i} \cdot \mathbf{pk}_{i,j} - \boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{d}^{\mathsf{v}}_{j,i} \cdot G \tag{2}$$

but instead checks whether $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G = \mathbf{R}_{i,j}$ and $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G = \mathbf{pk}_{i,j}$ and $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$, as specified in step 11 of $\mathcal{S}_{\mathsf{ECDSA}}$.

Because the code of $\mathcal{F}_{\mathsf{RVOLE}}$ enforces that

$$\boldsymbol{\chi}_{j,i} \cdot \mathbf{a}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} + \mathbf{d}^{\mathsf{u}}_{j,i} \qquad \text{and} \qquad \boldsymbol{\chi}_{j,i} \cdot \mathbf{a}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} + \mathbf{d}^{\mathsf{v}}_{j,i}$$

we know that if the consistency checks evaluated in $\mathcal{H}_2$ pass, then the checks in $\mathcal{H}_1$ also pass. We also know that if $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G = \mathbf{R}_{i,j}$ and $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G = \mathbf{pk}_{i,j}$, then the checks in both hybrids pass if and only if $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$. Thus, the adversary can only distinguish the two by setting $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G \neq \mathbf{R}_{i,j}$ or $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G \neq \mathbf{pk}_{i,j}$ for some corrupt $\mathcal{P}_j$ and contriving to pass the consistency check in $\mathcal{H}_1$, while failing the check (with certainty) in $\mathcal{H}_2$. Because $\mathbf{R}_{i,j}$ is fixed, there is exactly one value of $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}$ that will satisfy equation 1 for any assignment of $\boldsymbol{\chi}_{j,i}$ and $\mathbf{d}^{\mathsf{u}}_{j,i}$. Since $\boldsymbol{\chi}_{j,i}$ and $\mathbf{d}^{\mathsf{u}}_{j,i}$ are uniformly sampled and information-theoretically hidden from the adversary at the time that it must commit to $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}$, the probability that the adversary sends this value is exactly

$1/q$ if $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G \neq \mathbf{R}_{i,j}$. A similar argument implies that the adversary has a $1/q$ probability of satisfying equation 2 if $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G \neq \mathbf{pk}_{i,j}$.

Note that if a consistency check fails, the honest party that observes the failure will never again allow a signing session to produce an output when it involves the party that caused the failure. Even if an unbounded environment were permitted to invoke an unbounded number of signing sessions, at most $(t-1)\cdot(n-t+1)$ failed consistency checks can occur before there are no honest parties that are willing to sign with any corrupt party. The probability that at least one of the first $(t-1)\cdot(n-t+1)$ distinguishing attempts will result in success is upper-bounded by $(t-1)\cdot(n-t+1)/q \leq n^2/q$, and since $n = \nu(\lambda)$ is polynomially-bounded while q is exponential in λ, it follows that $\mathcal{H}_2 \approx_{\mathrm{s}} \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. This hybrid behaves identically to $\mathcal{H}_2$, except that if the environment triggers a single signing instance with ID sigid among a group of *exclusively* honest parties with messages that have consistent images under $\mathsf{SHA2}$, then the protocol code no longer runs. Instead, upon $(\mathtt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ on behalf of all of the parties, $\mathcal{S}$ locally evaluates $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$ and outputs $(\mathtt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to the environment on behalf of all parties.

Observe that in $\mathcal{H}_2$, a group of honest parties compute their views such that

$$
\begin{aligned}
r_i &\leftarrow \mathbb{Z}_q \quad \text{for every } i \in \mathbf{P} && (3)\\
r_i \cdot \boldsymbol{\chi}_{j,i} &= \mathbf{c}^{\mathsf{u}}_{i,j} + \mathbf{d}^{\mathsf{u}}_{j,i} \quad \text{for every } i,j \in \mathbf{P} : i \neq j\\
\mathsf{sk}_i \cdot \boldsymbol{\chi}_{j,i} &= \mathbf{c}^{\mathsf{v}}_{i,j} + \mathbf{d}^{\mathsf{v}}_{j,i} \quad \text{for every } i,j \in \mathbf{P} : i \neq j\\
\boldsymbol{\psi}_{j,i} &= \phi_j - \boldsymbol{\chi}_{j,i} \quad \text{for every } i,j \in \mathbf{P} : i \neq j\\
u &= \sum_{i \in \mathbf{P}} \left(r_i \cdot \phi_i + \sum_{j \in \mathbf{P} \setminus \{i\}} (r_i \cdot \boldsymbol{\psi}_{j,i} + \mathbf{c}^{\mathsf{u}}_{i,j} + \mathbf{d}^{\mathsf{u}}_{i,j}) \right) = r \cdot \phi\\
v &= \sum_{i \in \mathbf{P}} \left(\mathsf{sk}_i \cdot \phi_i + \sum_{j \in \mathbf{P} \setminus \{i\}} (r_i \cdot \boldsymbol{\psi}_{j,i} + \mathbf{c}^{\mathsf{v}}_{i,j} + \mathbf{d}^{\mathsf{v}}_{i,j}) \right) = \mathsf{sk} \cdot \phi\\
s &= \frac{\mathsf{SHA2}(m) \cdot \phi + r^{\mathsf{x}} \cdot v}{u} = \frac{\mathsf{SHA2}(m) + r^{\mathsf{x}} \cdot \mathsf{sk}}{r} && (4)
\end{aligned}
$$

The consistency checks introduced in $\mathcal{H}_2$ trivially pass when all of the participants are honest, and by inspection we can see that equations 3 and 4 yield a signature with a distribution identical to that produced by $\mathsf{ECDSASign}$, which implies that the verification check in step 10 of π_{ECDSA} always passes when all signing parties are honest. Thus the output distributions for all signing parties are identical in $\mathcal{H}_3$ and $\mathcal{H}_2$, and in both hybrids the probability of a failed signature is zero. No other values are observable by the adversary, and so the two hybrids are perfectly indistinguishable.

Hybrid $\mathcal{H}_4$. This hybrid behaves identically to $\mathcal{H}_3$, except when the environment triggers a single signing instance with ID sigid between a group containing two or more honest parties, but uses inconsistent messages with the honest parties. Suppose $\mathcal{P}_h$ is the last honest party in the group to be activated by the environment. In $\mathcal{H}_4$, $\mathcal{S}$ replaces m_j with m_h in the calculations of every honest

$\mathcal{P}_j$ for $j \in \mathbf{H} \setminus \{h\}$. If there exists some $j \in \mathbf{H} \setminus \{h\}$ such that environment sends $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m_j)$ to $\mathcal{P}_j$ and $\mathsf{SHA2}(m_j) \neq \mathsf{SHA2}(m_h)$, then $\mathcal{S}$ samples $w_h \leftarrow \mathbb{Z}_q$ instead of calculating w_h per the instructions in step 8 of π_{ECDSA} as in $\mathcal{H}_3$, always outputs $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to the environment on behalf of all honest parties, and ignores all future messages with the same sigid.

In $\mathcal{H}_3$, honest parties fail if they do not receive a valid signature as output, and we have argued in the context of $\mathcal{H}_3$ that a group of signers always receives a valid signature as output when their messages have the same image under $\mathsf{SHA2}$ and nobody deviates from the protocol. In $\mathcal{H}_3$, $\mathcal{S}$ always calculates $w_h := \mathsf{SHA2}(m_h) \cdot \phi_h + r^{\mathsf{x}} \cdot v_h$ and $w_j := \mathsf{SHA2}(m_j) \cdot \phi_j + r^{\mathsf{x}} \cdot v_j$. We make three observations. First, the leftmost terms of these equations are the *only* constituent parts of the final signature that depend upon m_h or m_j. Second, $\mathcal{S}$ effectively samples w_h and w_j uniformly subject to a condition on their sum, because v_j and v_h depend linearly on $\mathbf{c}^{\mathsf{v}}_{j,h} + \mathbf{d}^{\mathsf{v}}_{j,h}$ and $\mathbf{c}^{\mathsf{v}}_{h,j} + \mathbf{d}^{\mathsf{v}}_{h,j}$ respectively, and the latter values are sampled uniformly subject to a condition on their sum. Third, due to similar linear dependencies upon $\mathbf{c}^{\mathsf{u}}_{j,h} + \mathbf{d}^{\mathsf{u}}_{j,h}$ and $\mathbf{c}^{\mathsf{u}}_{h,j} + \mathbf{d}^{\mathsf{u}}_{h,j}$, we can conclude that u_j and u_h commit $\mathcal{S}$ to the *sum* of ϕ_j and ϕ_h, but $\mathcal{S}$ still has a degree of freedom in choosing the individual values.[5]

Summing and rewriting, we have

$$w_h + w_j = \mathsf{SHA2}(m_h) \cdot (\phi_h + \phi_j) + (\mathsf{SHA2}(m_j) - \mathsf{SHA2}(m_h)) \cdot \phi_j + r^{\mathsf{x}} \cdot (v_h + v_j)$$

If $\mathsf{SHA2}(m_h) = \mathsf{SHA2}(m_j)$, then $(\mathsf{SHA2}(m_j) - \mathsf{SHA2}(m_h)) \cdot \phi_j = 0$ and the signature is valid if the corrupt parties follow the protocol. If $\mathsf{SHA2}(m_h) \neq \mathsf{SHA2}(m_j)$, then $(\mathsf{SHA2}(m_j) - \mathsf{SHA2}(m_h)) \cdot \phi_j$ is distributed uniformly, because ϕ_j is, as we have observed, uniformly sampled and information-theoretically hidden from the adversary. All other terms that the honest parties contribute to the signature *are the same in either case*. In other words, if $\mathsf{SHA2}(m_h) \neq \mathsf{SHA2}(m_j)$, then w_h and w_j are not uniform subject to a condition on their sum, but simply uniform. This implies that the joint distribution of w_i for every $i \in \mathbf{H}$ is identical in $\mathcal{H}_4$ and $\mathcal{H}_3$, both when $\mathsf{SHA2}(m_h) = \mathsf{SHA2}(m_j)$ and when $\mathsf{SHA2}(m_h) \neq \mathsf{SHA2}(m_j)$.

Once the message, public key, and nonce are fixed, there is exactly one valid ECDSA signature. When $\mathsf{SHA2}(m_h) \neq \mathsf{SHA2}(m_j)$ and w_h and w_j are uniform without constraint, the chance that the resulting signature will be valid for any honest party's message (and that party will consequently output a signature in $\mathcal{H}_3$) is no greater than t/q in each signing session. Since $t < n = \nu(\lambda)$ is polynomial in λ, and we have assumed the number of signing sessions to be bounded by $\mu(\lambda)$, which is polynomial in λ, but q is exponential in λ, we can conclude that the distribution of honest party failures in $\mathcal{H}_4$ is statistically indistinguishable from the distribution in $\mathcal{H}_3$. It follows that $\mathcal{H}_4 \approx_{\mathrm{s}} \mathcal{H}_3$ overall. For the remainder of this proof, we will assume that if any group of *honest* signing parties does not output a failure, then their messages have identical images under $\mathsf{SHA2}$.

[5] Fixing one of these two values also fixes the other, but the simulator cannot calculate both without implicitly breaking the discrete logarithm problem on R. Fortunately it will not be necessary to calculate both.

Hybrid $\mathcal{H}_5$. The behavior of this hybrid differs from $\mathcal{H}_4$ when the environment triggers a signing instance among a group of parties, some of whom are corrupt. In $\mathcal{H}_5$, if the honest parties receive messages that have identical images under $\mathsf{SHA2}$, then $\mathcal{S}$ uses the $\mathsf{ECDSASign}$ to generate the signature, and embeds it into the protocol by altering the code of one of the honest signers. Specifically, $\mathcal{S}$ follows the code of the honest parties on their behalves until step 7 of π_{ECDSA}. Whichever honest party reaches this step *last* is designated $\mathcal{P}_h$ (as before), and if the honest parties have messages with identical images under $\mathsf{SHA2}$, then the code of $\mathcal{P}_h$ is replaced for the remainder of the protocol.

When the time comes for $\mathcal{S}$ to decommit $\mathcal{P}_h$'s contribution to the nonce on behalf of $\mathcal{F}_{\mathsf{Com}}$, rather that decommitting the value of R_h that was committed by $\mathcal{P}_h$ in step 6 of π_{ECDSA}, $\mathcal{S}$ instead samples $(s, r^{\mathsf{x}}) \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$, reconstructs R from the x-coordinate r^{x}, computes

$$R_h := R - \sum_{k \in \mathbf{P}^{\text{-}h}} R_k$$

and then decommits this value of R_h on behalf of $\mathcal{F}_{\mathsf{Com}}$. This embeds the value of r^{x} that was sampled by $\mathsf{ECDSASign}$ into the protocol output, if the parties do not deviate from the protocol. Note that the distribution of r^{x} has not changed: in both $\mathcal{H}_5$ and $\mathcal{H}_4$ it is uniform.

Next, if the consistency checks (specified in step 11 of $\mathcal{S}_{\mathsf{ECDSA}}$) pass, then $\mathcal{S}$ uses its knowledge of the corrupt parties' inputs and outputs from $\mathcal{F}_{\mathsf{RVOLE}}$ to predict the values of u_i and w_i that all of the parties apart from $\mathcal{P}_h$ would use, if no parties cheated. We will denote the *sum* of these predicted values as $\hat{u}$ and $\hat{w}$ respectively. These values depend upon ϕ_i for $i \in \mathbf{C}$, which might have been used inconsistently in interactions with the different honest parties, if the corrupt parties have misbehaved. $\mathcal{S}$ defines the true value of ϕ_i to be the value implied by the interaction between $\mathcal{P}_i$ and $\mathcal{P}_h$. Note that $\phi_i - \boldsymbol{\psi}_{i,h} - \boldsymbol{\chi}_{i,h} = 0$ by definition, which implies that there is no discrepancy between the values $\mathcal{P}_h$ computes if the corrupt parties follow the protocol and the values it computes if they misbehave, conditioned on the fact that the consistency check passes.

If we let δ_j^{u} for $j \in \mathbf{H} \setminus \{h\}$ represent the sum of the terms comprising u_j that arise from interactions between the honest $\mathcal{P}_j$ and the other honest parties, and likewise let δ_j^{v} represent the sum of the honestly-derived terms comprising v_j, then we have

$$\delta_j^{\mathsf{u}} = r_j \cdot \phi_j + \sum_{k \in \mathbf{H} \setminus \{j\}} (\mathbf{c}_{j,k}^{\mathsf{u}} + \mathbf{d}_{j,k}^{\mathsf{u}}) \qquad \text{for } j \in \mathbf{H}$$

$$\delta_j^{\mathsf{u}} = \mathsf{sk}_j \cdot \phi_j + \sum_{k \in \mathbf{H} \setminus \{j\}} (\mathbf{c}_{j,k}^{\mathsf{v}} + \mathbf{d}_{j,k}^{\mathsf{v}}) \qquad \text{for } j \in \mathbf{H}$$

$$\hat{u} := \sum_{i \in \mathbf{C}} \left(\mathbf{a}_{i,h}^{\mathsf{u}} \cdot \Big(\sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + \boldsymbol{\psi}_{h,i} \Big) + \mathbf{c}_{i,h}^{\mathsf{u}} + \mathbf{d}_{i,h}^{\mathsf{u}} \right)$$

$$+ \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta_j^{\mathsf{u}} + r_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right)$$

$$\hat{v} := \sum_{i \in \mathbf{C}} \left(\mathbf{a}_{i,h}^{\mathsf{v}} \cdot \Big(\sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + \boldsymbol{\psi}_{h,i} \Big) + \mathbf{c}_{i,h}^{\mathsf{v}} + \mathbf{d}_{i,h}^{\mathsf{v}} \right) + \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta_j^{\mathsf{v}} + \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right)$$

$$\hat{w} := \mathsf{SHA2}(m) \cdot \sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + r^{\mathsf{x}} \cdot \hat{v}$$

Note that because $\mathbf{c}^{\mathsf{u}}$, $\mathbf{d}^{\mathsf{u}}$, $\mathbf{c}^{\mathsf{v}}$, and $\mathbf{d}^{\mathsf{v}}$ are uniformly sampled subject to constraints upon their component-wise sums, δ_j^{u} and δ_j^{v} are uniform when considered indepently of the view of $\mathcal{P}_h$. By the same argument as we made in the context of $\mathcal{H}_3$, these constraints imply that the output of the protocol in $\mathcal{H}_4$ is a valid ECDSA signature on m under pk and R when all parties follow the protocol.

In $\mathcal{H}_5$, $\mathcal{S}$ samples $u_h \leftarrow \mathbb{Z}_q$ and $\delta_j^{\mathsf{u}} \leftarrow \mathbb{Z}_q$ and $\delta_j^{\mathsf{v}} \leftarrow \mathbb{Z}_q$ for $j \in \mathbf{H} \setminus \{h\}$ uniformly, calculates $\hat{u}$ and $\hat{w}$ as defined above, and then computes

$$w_h := s \cdot u_h + \sum_{k \in \mathbf{P}^{\text{-}h}} \Big(s \cdot \hat{u}_k - \hat{w}_k \Big)$$

This embeds the value of s that was sampled by ECDSASign into the protocol output, if the parties do not deviate from the protocol. In both hybrids u_j and w_j for $j \in \mathbf{H}$ are all uniformly distributed subject to the fact that s is the single valid ECDSA signature on m that exists under pk and R when no party deviates (and the honest parties have messages with identical images under $\mathsf{SHA2}$). If the corrupt parties *do* deviate then the offsets they induce upon the output satisfy the same algebraic relationship with the embedded value of s in $\mathcal{H}_5$ as they do with the hypothetical value of s that would occur if they did not cheat in $\mathcal{H}_4$. Thus $\mathcal{H}_5 = \mathcal{H}_4$.

Hybrid $\mathcal{H}_6$. This final hybrid differs from $\mathcal{H}_5$ in the following way: $\mathcal{S}$ no longer acts on behalf of any honest parties, nor does it use ECDSAGen or ECDSASign internally to sample signatures. Instead, $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ is fully implemented in $\mathcal{H}_6$ (that is, $\mathcal{S} = \mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$), and the experiment now incorporates $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$. The honest parties run dummy-party code as is standard for ideal-world experiements in the UC model, and $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ speaks to $\mathcal{F}_{\mathsf{ECDSA}}$ on behalf of corrupt parties.

The differences between $\mathcal{H}_6$ and $\mathcal{H}_5$ are purely syntactical, which is to say that $\mathcal{H}_5 = \mathcal{H}_6$. Notice that in $\mathcal{H}_5$, $\mathcal{S}$ did not require knowledge of r_h or sk_h in order to simulate. Moreover, because sk_k for $k \in \mathbf{P}$ were computed by adding a uniform secret-sharing of zero ζ_k to the interpolated Shamir-shares $\mathsf{lagrange}(\mathbf{P}, k, 0) \cdot p(k)$ of the secret key, sk_k for $k \in \mathbf{P}$ were uniform subject to

$$\sum_{k \in \mathbf{P}} \mathsf{sk}_k \cdot G = \mathsf{pk}$$

In $\mathcal{H}_6$, at initialization time, $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of the corrupt parties, and learns pk but *not* the discrete logarithm of pk. At signing time, it samples sk_j for $j \in \mathbf{H} \setminus \{h\}$ uniformly, and computes pk_h such that the above equation holds. $\mathcal{S}$ waits to invoke $\mathcal{F}_{\mathsf{ECDSA}}$ on behalf of the corrupt parties until after $\mathcal{F}_{\mathsf{ECDSA}}$ is invoked by the very last honest party $\mathcal{P}_h$, and uses m_h (the message on which a signature was requested by $\mathcal{P}_h$) on the corrupt parties' behalves. If $\mathcal{S}$ receives (s, r^{x}) from $\mathcal{F}_{\mathsf{ECDSA}}$ on the corrupted parties' behalves, then it embeds these values into the protocol just as it did in $\mathcal{H}_5$. If it receives a failure message from $\mathcal{F}_{\mathsf{ECDSA}}$, then it samples (s, r^{x}) uniformly amd embeds them, just as it did in $\mathcal{H}_5$. Since the failure conditions are identical, pk_j for $j \in \mathbf{H}$ are identically distributed, the signature values are identically distributed, and the embedding of the signature is otherwise performed identically in the two hybrids, $\mathcal{H}_6 = \mathcal{H}_5$.

Since we now have

$$\mathcal{H}_6 = \left\{ \textsc{Ideal}_{\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{Z}}(\lambda, z) : \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda) \right\}_{\substack{\lambda\in\mathbb{N}, n\in[2,\nu(\lambda)],\\ t\in[2,n], z\in\{0,1\}^*}}$$

and by transitivity we also have

$$\mathcal{H}_6 \approx_{\mathrm{s}} \mathcal{H}_0 = \left\{ \textsc{Real}_{\pi_{\mathsf{ECDSA}}(\mathcal{G},n,t),\mathcal{A},\mathcal{Z}}(\lambda, z) : \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda) \right\}_{\substack{\lambda\in\mathbb{N}, n\in[2,\nu(\lambda)],\\ t\in[2,n], z\in\{0,1\}^*}}$$

we can conclude that Theorem 4.1 holds. □

5 Cost Analysis

In this section, we give a closed-form accounting of the bandwidth costs of our protocol and its various building blocks. We also account for the number of elliptic curve scalar operations that our protocol requires during signing, since this is the dominant computational cost. We begin with all of the non-VOLE building blocks that were introduced in section 3.1; in each case we assume use of the realization that was suggested in that section. To wit, we assume that an $\mathcal{F}_{\mathsf{Com}}$ commitment to any payload requires $2\lambda_{\mathsf{c}}$ bits to be transmitted, and a decommitment to a payload of size x requires $2\lambda_{\mathsf{c}} + x$ bits. We assume $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$ requires a one-time setup cost of $(t-1)$ commitments and decommitments to λ_{c} bits on the part of each party, and that invocation by that set of parties is free in terms of bandwidth thereafter. The cost of realizing $\mathcal{F}_{\mathsf{DLKeyGen}}(\mathcal{G}, n, t)$ via the DKLs protocol [DKLs19] has already been reported by Doerner et al. [DKL+23]. We assume a mild optimization of their protocol, where all commitments and duplicate values transmitted by any party to another party are coalesced, and arrive at a bandwidth cost of

$$\begin{aligned}&\mathsf{KeyGenCost}(n, \lambda_{\mathsf{c}}, \kappa, |G|) \mapsto \\ &\quad (n-1) \cdot (4\lambda_{\mathsf{c}} + (\kappa + |G|) \cdot \lceil \lambda_{\mathsf{c}} / \log_2 \lambda_{\mathsf{c}} \rceil + \kappa + |G|)\end{aligned}$$

In addition to the foregoing costs, the VOLE instantiations described in the next two subsections both rely upon oblivious transfer. We suggest to realize this via the OT-extension protocol of Roy [Roy22], with base OTs supplied by the endemic OT protocol of Zhou et al. [ZZZR23]. We instantiate the latter primitive over the same group $\mathcal{G}$ in which signatures are to be computed. Zhou et al. claim that for ℓ_{OT} OT instances their protocol has an average per-party bandwidth cost of

$$\mathsf{OTCost}(\lambda_{\mathsf{c}}, \kappa, |G|, \ell_{\mathsf{OT}}) \mapsto \frac{\lambda_{\mathsf{c}} + (2\ell_{\mathsf{OT}} + 1) \cdot |G| + ((2\ell_{\mathsf{OT}} + 1) \cdot |G| + (3\ell_{\mathsf{OT}} + 1) \cdot \kappa) \cdot \lceil \lambda_{\mathsf{c}} / \log_2 \lambda_{\mathsf{c}} \rceil}{2}$$

In order to meet our round-count requirements, Roy's OT-extension protocol must be modified via the Fiat-Shamir heuristic to run in two rounds. In the first round, the OT receiver sends a message to the OT sender, and in the second round, the sender replies. Roy's protocol requires a one-time setup that comprises exactly λ_{c} instances of OT. This is followed by any number of extension batches. Per the accounting of Doerner et al. [DKL+23], each batch of ℓ_{OTE} *random* OT instances has an average per-party bandwidth cost of

$$\mathsf{ROTECost}(\lambda_{\mathsf{c}}, \ell_{\mathsf{OTE}}) \mapsto \left(\frac{3}{2} + \frac{1}{2k_{\mathsf{SSOT}}}\right) \cdot ({\lambda_{\mathsf{c}}}^2 + \lambda_{\mathsf{c}}) + \frac{\lambda_{\mathsf{c}} \cdot \ell_{\mathsf{OTE}}}{2k_{\mathsf{SSOT}}}$$

where k_{SSOT} is a parameter that also impacts computation time. Roy suggested that $k_{\mathsf{SSOT}} = 2$ yields a strict improvement upon all other OT-extension protocols, and we follow this advice when calculating concrete costs. If correlated OT-extension instances are required then

$$\mathsf{COTECost}(\lambda_{\mathsf{c}}, \ell_{\mathsf{OTE}}, |m|) \mapsto \ell_{\mathsf{OTE}} \cdot |m|/2 + \mathsf{ROTECost}(\lambda_{\mathsf{c}}, \ell_{\mathsf{OTE}})$$

where $|m|$ is the size of the message to be transmitted.

5.1 An Optimized DKLs VOLE

In this section, we describe a simple optimization of the DKLs-derived VOLE protocol introduced in section 3.1. Recall that our protocol is based upon the multiplication protocol of Doerner et al. [DKLs19]. In this subsection (and only this subsection) we will use their notation, except for security parameters (in which case, we use λ_{c} and λ_{s} for the computational and statistical security parameters, respectively), and their parameter ℓ, which we rename to ℓ_{DKLs} in order to avoid collision with our parameter ℓ. Recall also that we have eliminated their batching technique (i.e. hardcoded $\ell_{\mathsf{DKLs}} = 1$), hardcoded Bob's adjustment message $\gamma_{\mathsf{B}} = 0$, and applied the "forced-reuse" technique from Chen et al. [CCD+20] to Bob's side of the protocol, in order to tranform it into a multiplication protocol into a random vector OLE protocol. The last of these adjustments introduces a new vector-length parameter ℓ for Alice's input, and for the purposes of the optimization we present here, this new parameter ℓ

must be treated similarly to their parameter ℓ_{DKLs}. In Steps 5 and 6 of Protocol 1 of their paper (after our adjustments), Alice computes

$$\mathbf{r} \coloneqq \left\{ \{\tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{z}}_{\mathsf{A},j,i} + \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{z}}_{\mathsf{A},j,i}\}_{i\in[\ell]} \right\}_{j\in[\kappa+2\lambda_\mathsf{s}]}$$
$$\mathbf{u} \coloneqq \{\tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{a}}_i + \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{a}}_i\}_{i\in[\ell]}$$

and sends $\mathbf{r}$ and $\mathbf{u}$ to Bob, who then aborts if

$$\bigvee_{\substack{i\in[\ell],\\ j\in[\kappa+2\lambda_\mathsf{s}]}} \mathbf{r}_{j,i} + \tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{z}}_{\mathsf{B},j,i} + \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{z}}_{\mathsf{B},j,i} \neq \boldsymbol{\beta}_j \cdot \mathbf{u}_i$$

Rewriting the above equation, we have

$$\bigvee_{\substack{i\in[\ell],\\ j\in[\kappa+2\lambda_\mathsf{s}]}} \mathbf{r}_{j,i} \neq \boldsymbol{\beta}_j \cdot \mathbf{u}_i - \tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{z}}_{\mathsf{B},j,i} - \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{z}}_{\mathsf{B},j,i}$$

and we observe that in the non-programmable global random oracle, the following optimization is possible: Instead of sending $\mathbf{r}$, Alice computes $\tilde{r} \coloneqq \mathsf{RO}(\mathsf{sid}, \mathbf{r})$ and transmits $\tilde{r}$ with $\mathbf{u}$. Bob aborts if

$$\tilde{r} \neq \mathsf{RO}\left(\mathsf{sid}, \{\boldsymbol{\beta}_j \cdot \mathbf{u}_i - \tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{z}}_{\mathsf{B},j,i} - \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{z}}_{\mathsf{B},j,i}\}_{i\in[\ell],j\in[\kappa+2\lambda_\mathsf{s}]}\right)$$

Note that the above optimization involves Alice sending strictly less information to Bob. Intuitively, other than the event that a collision is found for the random oracle, Alice will not be able to pass the check unless her input to RO exactly matches Bob's input to RO, which in turn is a simple rearrangement of the equation that he must verify in the original protocol. This holds trivially in the non-programmable global random oracle, which is already required by the protocol of Doerner et al., and the proof of security is only slightly less trivial if RO replaced by a collision-resistant hash function.

The unmodified protocol involves a correlated OT-extention batch of size $\ell_{\mathsf{OTE}} = \kappa + 2\lambda_\mathsf{s}$, where each element of the batch has a payload of size $2\kappa \cdot \ell$ bits, and the transmission of $\ell \cdot (\kappa + 2\lambda_\mathsf{s} + 1)$ additional elements from $\mathbb{Z}_q$ directly from Alice to Bob. This optimization reduces the latter figure to just one $\mathbb{Z}_q$ element. This brings the total online (i.e. excluding one-time setup) average per-party bandwidth cost of our DKLs-derived VOLE protocol to

$$\mathsf{VOLECost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, \ell) \mapsto \mathsf{COTECost}(\lambda_\mathsf{c}, \kappa + 2\lambda_\mathsf{s}, 2\kappa \cdot \ell) + \kappa \cdot (\ell + 1)/2$$

The one-time setup for our protocol comprises the one-time setup for correlated OT-extensions, plus the sending of a single security-parameter-length seed from Bob to Alice. Thus, assuming Roy's OT-extension protocol is used, we have an average per-party bandwidth cost of

$$\mathsf{VOLESetupCost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, |G|) \mapsto \mathsf{OTCost}(\lambda_\mathsf{c}, \kappa, |G|, \lambda_\mathsf{c}) + \lambda_\mathsf{c}/2$$

5.2 VOLE from HMRT22

Next, we present the cost of using an alternate VOLE protocol derived from the work of Haitner et al. [HMRT22]. We remind the reader that their protocol realizes a weaker functionality than the one we have specified, and we have *not* proven the combination secure, and we remind the reader that the protocol of Haitner et al. requires an additional round, relative to the DKLs-derived VOLE presented in section 5.1.

In order to use their protocol, it is necessary to perform a one-time setup for correlated OT-extensions. Thus

$$\mathsf{VOLESetupCost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, |G|) \mapsto \mathsf{OTCost}(\lambda_\mathsf{c}, \kappa, |G|, \lambda_\mathsf{c}) + \lambda_\mathsf{c}/2$$

The evaluation stage of their protocol involves a batch of $\ell_{\mathsf{OTE}} = \kappa + 4\lambda_\mathsf{s}$ correlated OT-extensions. Per a random-oracle-based optimization mentioned in their paper, the only additional data that must be sent is a single λ_c-bit seed, plus a single element of $\mathbb{Z}_q$.[6] If a VOLE protocol is derived from their protocol via a similar "forced-reuse" technique to the one we employ when modifying the DKLs protocol, then the average per-party bandwidth cost is

$$\mathsf{VOLECost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, \ell) \mapsto \mathsf{COTECost}(\lambda_\mathsf{c}, \kappa + 4\lambda_\mathsf{s}, \kappa \cdot \ell) + (\kappa \cdot \ell + \lambda_\mathsf{c})/2$$

5.3 Our ECDSA Protocol

The one-time initialization for π_{ECDSA} involves running $\mathcal{F}_{\mathsf{DLKeyGen}}$ (which is realized by π_{DLKeyGen} [DKL+23]) and initializing two instances of $\mathcal{F}_{\mathsf{RVOLE}}$ per pair of parties. Each party must also commit and release a pair of λ_c-bit seeds, in order to initialize the protocol that realizes $\mathcal{F}_{\mathsf{Zero}}$. Coalescing these commitments with the ones required by π_{DLKeyGen} yields an average per-party bandwidth cost of

$$\begin{aligned}&\mathsf{SetupCost}(n, \lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, |G|) \mapsto \\ &\quad (n-1) \cdot \begin{pmatrix} 5\lambda_\mathsf{c} + (\kappa + |G|) \cdot \lceil \lambda_\mathsf{c} / \log_2 \lambda_\mathsf{c} \rceil + \kappa + |G| \\ + 2 \cdot \mathsf{VOLESetupCost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, |G|) \end{pmatrix}\end{aligned}$$

Our signing protocol is very simple. Each pair of parties performs two VOLE evaluations, and each party $\mathcal{P}_i$ commits and releases R_i and transmits $\mathbf{\Gamma}^{\mathsf{u}}_{i,j}$, $\mathbf{\Gamma}^{\mathsf{v}}_{i,j}$, $\boldsymbol{\psi}_{i,j}$, pk_i, w_i, and u_i to every $\mathcal{P}_j$ such that $j \neq i$. This gives us a total average per-party bandwidth cost of

$$\begin{aligned}&\mathsf{SignCost}(t, \lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, |G|) \mapsto \\ &\quad (t-1) \cdot (4\lambda_\mathsf{c} + 3\kappa + 4|G| + 2 \cdot \mathsf{VOLECost}(\lambda_\mathsf{c}, \lambda_\mathsf{s}, \kappa, 2))\end{aligned}$$

Finally, each party must perform $6t - 2$ elliptic curve scalar operations in order to generate a signature.

[6] They specify only that the random oracle can be used to compress the value denoted in their paper as $\mathbf{v}$, but do not give specifics. We assume that a seed is used to generate a random vector, and the single $\mathbb{Z}_q$ element is used to adjust that vector such that it meets the constraints that they require.

5.4 Concrete Results

In table 1, we substitute values into the above equations to derive the concrete average per-party bandwidth costs for common security parameters. We assume that point compression is used for elements of $\mathbb{G}$, such that they require only one byte more than elements of $\mathbb{Z}_q$.[7] In all cases, we assume that $\kappa = 2\lambda_c$ and $\lambda_s = 80$. Note that since the seeds used to initialize the VOLE protocol and the protocol that realized $\mathcal{F}_{\mathsf{Zero}}$ can be combined, the cost of VOLE setup (and therefore the overall setup cost) is the same regardless of which VOLE method is used.

For comparison, when $\lambda_c = 256$ and $\lambda_s = 80$, the 2-of-n signing protocol of Doerner et al. [DKLs18] requires each party to send 116.4 KiB (on average), whereas our new protocol (with the DKLs-derived VOLE) requires only 59.7 KiB per party to be sent. On the other hand, our new protocol has the same communication pattern as theirs (under pipelining), requires fewer elliptic curve scalar operations than theirs does,[8] realizes a standard functionality, whereas their functionality allows the adversary to bias R, and achieves statistical security, whereas theirs is secure only assuming that the computational Diffie-Dellman problem is hard in $\mathbb{G}$ and that ECDSA is a signature scheme over $\mathcal{G}$. We therefore claim that our protocol is strictly superior to the original 2-of-n DKLs protocol.

The t-of-n protocol of Doerner et al. [DKLs19] requires $(t-1)\cdot 88.3$ KiB to be sent by each party (on average) when the $(\lceil \log_2(t) \rceil + 6)$-round variant is used.[9] Our new protocol requires only $(t-1)\cdot 59.7$ KiB to be sent, has only three rounds (or two, if pipelining is employed), and achieves statistical security, whereas theirs is secure assuming that the computational Diffie-Hellman problem is hard in $\mathbb{G}$. However, their protocol requires each party to compute only 6 elliptic curve scalar operations during signing, and ours requires $6t - 2$. Given the efficiency of elliptic curve operations on modern hardware, and the additional latency incurred by additional parties, we believe our new protocol to have the advantage in nearly any real-world deployment scenario.

6 A Two-Round Protocol for Honest Majorities

When the number of corrupt parties is strictly less than $t/2$, a much simpler protocol is possible than the one presented in section 3, leveraging honest-majority techniques for significant bandwidth and round-efficiency improvements. In spite of its simplicity, we present it here for the sake of completeness, and give a provisional theorem.

[7] This is not true of elliptic curves in general, but is true of the ones over which ECDSA is most commonly deployed.

[8] While their protocol only requires 9 such operations as implemented, achieving UC-security for their protocol requires a straight-line extractable proof of knowledge [Fis05, Ks22], which requires many more.

[9] Their 10-round variant requires more bandwidth, but they do not give a precise figure.

λ_c	128	192	256
κ	256	384	512
$\lvert G\rvert$	264	392	520
Setup	$(n-1)\cdot 3240768$	$(n-1)\cdot 9857712$	$(n-1)\cdot 21437968$
Signing (DKLs)	$(t-1)\cdot 488736$	$(t-1)\cdot 966560$	$(t-1)\cdot 1600032$
Signing (HMRT)	$(t-1)\cdot 367776$	$(t-1)\cdot 686816$	$(t-1)\cdot 1095968$

Table 1: Bandwidth costs in total bits transmitted per party, for t signers out of n total parties. Note that in all cases, the statistical parameter $\lambda_s = 80$.

Theorem 6.1 (Informal Honest-Majority Security Theorem)**.** *When* (t choose $\lceil t/2 \rceil$) $\in$ poly(λ)*, there exists a two-round protocol that UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *against a malicious adversary that statically corrupts fewer than* $t/2$ *parties, assuming the existence of pseudo-random functions.*

The protocol begins by sampling *replicated* secret shares of sk, r, $\zeta = 0$, and ϕ, with a reconstruction threshold of $\lceil t/2 \rceil$. To non-interactively generate replicated secret shares of zero, there is a direct extension of the protocol we have given for realizing $\mathcal{F}_{\mathsf{Zero}}$ in section 3.1. To non-interactively sample replicated secret shares of a *uniform* value, one can perform a similar trick: simply replicate shares of a seed, and use a PRF to expand the replicated seeds when necessary.

It is possible to perform multiplications of the shared values in replicated form; however, the output shares would also be replicated and therefore inefficient to send. Instead, the parties non-interactively convert their replicated sharings into Shamir sharings of degree $\lceil t/2 \rceil - 1$ via the technique of Cramer, Damgård, and Ishai [CDI05], and then perform a standard non-interactive multiplication (as in the BGW protocol [BGW88], without degree-reduction) to compute Shamir shares of u and v. The latter sharings are of degree $t-1$ if t is odd, or $t-2$ if t is even. Following this, a non-interactive linear combination yields Shamir shares of w.

The final honest-majority protocol, then, is two rounds: the parties perform all of the non-interactive operations specified above, and swap degree-$(\lceil t/2 \rceil - 1)$ shares shares of R over $\mathbb{G}$. They check that these shares lie on a polynomial of the correct degree, and if so, then they interpolate R and use $r^{\times}$ to compute shares of w, which they swap. These are interpolated and the signature assembled and verified.

Honest-majority techniques make our consistency check and the commit-and-release mechanism for R superfluous. Since the honest parties' shares fully specify R, any attempt by the adversary to bias this value will result in a polynomial of incorrect degree, which can be detected. Since the multiplication operations are completely non-interactive, any cheating on the part of the adversary *must* be independent of the honest parties' secrets, and therefore expressable in terms of simple linear offsets relative to the expected values, which can be perfectly detected by verifying the signature, just as in the protocol from

section 3. In terms of communication, each party must only send a single share of R to the others, followed by one share each of u and w; thus, when $\kappa = 256$, the total amount of data sent by every party to each of the others is 776 bits.

Note that in the above scheme, the size of each replicated secret share is a factor of (t choose $\lceil t/2 \rceil$) greater than the size of an ordinary additive share. In the case that this yields impractically large shares, Shamir sharing can be used throughout the protocol, and sharings of zero and uniform values can be sampled interactively via the well-known techniques of Feldman [Fel87] and Pedersen [Ped91]. Under this modification, the protocol requires three rounds.

Acknowledgements

The authors of this work were supported by the NSF under grants 1646671, 1816028, and 2055568, by the ERC under projects NTSC (742754) and SPEC (803096), by ISF grant 2774/2, and by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

References

[ANO+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy, (S&P)*, 2022.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1989.

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In *Advances in Cryptology – CRYPTO 2022, part IV*, 2022.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, 1988.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In *Advances in Cryptology – CRYPTO 2020, part III*, 2020.

[CCL+23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939:78–104, 2023.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proceedings of the Second Theory of Cryptography Conference, TCC 2005*, 2005.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *Proceedings of the 27th ACM Conference on Computer and Communications Security, (CCS)*, 2020.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA)*, 2015.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

[DKL+23] Jack Doerner, Yashvanth Kondi, Eysa Lee, abhi shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *Proceedings of the 44th IEEE Symposium on Security and Privacy, (S&P)*, 2023.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *Proceedings of the 39th IEEE Symposium on Security and Privacy, (S&P)*, 2018.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *Proceedings of the 40th IEEE Symposium on Security and Privacy, (S&P)*, 2019.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS)*, 1987.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology – CRYPTO 2005*, 2005.

[GS22a] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. *IACR Cryptology ePrint Arch.*, 2022.

[GS22b] Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. In *Advances in Cryptology – EUROCRYPT 2022, part I*, 2022.

[HMRT22] Iftach Haitner, Nikolaos Makriyannis, Samuel Ranellucci, and Eliad Tsfadia. Highly efficient OT-based multiplication protocols. In *Advances in Cryptology – CRYPTO 2022, part I*, 2022.

[Ks22] Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In *Advances in Cryptology – ASIACRYPT 2022, part II*, 2022.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 25th ACM Conference on Computer and Communications Security, (CCS)*, 2018.

[Nat13] National Institute of Standards and Technology. FIPS PUB 186-4: Digital Signature Standard (DSS). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, 2013.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In *Advances in Cryptology – CRYPTO 2021, part I*, 2021.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT 1999*, 1999.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO 1991*, 1991.

[Roy22] Lawrence Roy. SoftSpokenOT: Communication-computation tradeoffs in OT extension. In *Advances in Cryptology – CRYPTO 2022, part I*, 2022.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology – CRYPTO 1989*, 1989.

[ZZZR23] Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. Endemic oblivious transfer via random oracles, revisited. In *Advances in Cryptology – CRYPTO 2023, part I*, 2023.