Threshold ECDSA from ECDSA Assumptions: The Multiparty Case

Jack Doerner
j@ckdoerner.net
Northeastern University

Yashvanth Kondi
ykondi@ccs.neu.edu
Northeastern University

Eysa Lee
eysa@ccs.neu.edu
Northeastern University

abhi shelat
abhi@neu.edu
Northeastern University

Abstract

Cryptocurrency applications have spurred a resurgence of interest in the computation of ECDSA signatures using threshold protocols—that is, protocols in which the signing key is secret-shared among n parties, of which any subset of size t must interact in order to compute a signature. Among the resulting works to date, that of Doerner et al. [DKLs18] requires the most natural assumptions while also achieving the best practical signing speed. It is, however, limited to the setting in which the threshold is two. We propose an extension of their scheme to *arbitrary* thresholds, and prove it secure against a malicious adversary corrupting up to one party less than the threshold under only the Computational Diffie-Hellman Assumption in the Global Random Oracle model, an assumption strictly weaker than those under which ECDSA is proven.

Whereas the best current schemes for threshold-two ECDSA signing use a Diffie-Hellman Key Exchange to calculate each signature's nonce, a direct adaptation of this technique to a larger threshold t would incur a round count linear in t; thus we abandon it in favor of a new mechanism that yields a protocol requiring $\lceil \log(t) \rceil + 6$ rounds in total. We design a new consistency check, similar in spirit to that of Doerner et al., but suitable for an arbitrary number of participants, and we optimize the underlying two-party multiplication protocol on which our scheme is based, reducing its concrete communication and computation costs.

We implement our scheme and evaluate it among groups of up to 256 of co-located and geographically-distributed parties, and among small groups of embedded devices. We find that in the LAN setting, our scheme outperforms all prior works by orders of magnitude, and that it is efficient enough for use even on smartphones or hardware tokens. In the WAN setting we find that, despite its logarithmic round count, our protocol outperforms the best constant-round protocols in realistic scenarios.

1 Introduction

Threshold Digital Signature Schemes [Des87] allow a group of individuals to delegate their joint authority to sign a message to any subcommittee among themselves that is larger than a certain, predetermined size. Specifically, a t-of-n threshold signature scheme is a set of protocols that allow n parties to jointly generate a single public key, along with n private shares of a joint secret key sk, and then securely sign messages if and only if t of those parties participate in the signing operation. In addition to the standard unforgeability properties required of all signature schemes, threshold schemes must satisfy the properties of privacy against $t-1$ malicious participants with respect to the secret key shares of honest parties, and correctness against $t-1$ malicious participants with respect to signature output. That is, no group of $t-1$ colluding parties should be able to recover the secret key, even by interacting with additional honest parties, nor should they be able to trick an honest party into signing a message unwillingly. Threshold signature schemes are thus best modeled as a special case of secure multiparty computation (MPC).

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a standardized [Nat13,Ame05,Bro10] derivative of the DSA scheme of David Kravitz [Kra93], which improves upon the efficiency of its forebear by replacing arithmetic modulo a prime with operations over an elliptic curve. It is widely deployed in various web-security technologies such as DNSSec and TLS, in various authentication protocols, in binary signing, and in crypocurrencies, including Bitcoin [Bit17] and Ethereum [Woo17]. Although ECDSA is in widespread use, designing threshold signing protocols for ECDSA has been challenging due to the unusual structure of the signing algorithm: in each signature, a nonce k, its multiplicative inverse $1/k$, and the product sk/k (where sk is the secret key) all appear simultaneously. Computing these values efficiently in the multiparty context is the primary difficulty that threshold schemes must address.

MacKenzie and Reiter [MR01] constructed a 2-of-2 ECDSA protocol using multiplicative sharings of k and sk, which allowed shares of sk/k and $1/k$ to be computed via local operations, but their protocol required a mechanism to verify that the shares have been computed correctly. For this, they employed additively homomorphic encryption. Gennaro et al. [GGN16] extended this technique, introducing a six-round protocol for general t-of-n signing, and Boneh et al. [BGG17] subsequently optimized their extension in terms of computational efficiency, and reduced the round count to four. Meanwhile Lindell [Lin17] introduced optimizations in the 2-of-2 setting, such that key-generation and signing required only 2.4 seconds and 37 milliseconds in practice, respectively. Unfortunately, these schemes all require expensive zero-knowledge proofs, as well as the use of Paillier Encryption [Pai99], which leads both to poor performance and to reliance upon assumptions such as the Decisional Composite Residuosity Assumption (and a new assumption about the Paillier cryptosystem, in Lindell's case) that are foreign to the mathematics on which ECDSA is based.

Doerner et al. [DKLs18] propose an alternative solution for 2-of-n threshold key generation and signing: while their protocol retains the multiplicative shar-

ings of prior approaches, they forgo operating on Paillier ciphertexts. Instead, they construct a new, hardened variant of Gilboa's multiplication-by-oblivious-transfer technique [Gil99], by which their protocol converts multiplicative shares into additive shares, and thereby produces additive shares of the final ECDSA signature. Security against malicious adversaries is achieved via a novel *consistency check* that leverages relationships among various elements of an ECDSA signature to ensure that the multipliers receive consistent inputs. Their scheme requires only two rounds and outperforms prior schemes by one to two orders of magnitude in terms of computational efficiency, such that signatures can be produced in under four milliseconds, and key generation for two parties can be completed in under 45 milliseconds. Moreover, their scheme was proven secure in the Random Oracle model using only the assumption that ECDSA is a signature scheme, and the Computational Diffie-Hellman Assumption [DH76] over the *same* curve as the signature itself uses. The latter assumption is native to the primitive on which ECDSA is based, and it is implied by the Generic Group Model [Sho97] (in which ECDSA is proven secure [Bro05]).

While the 2-of-n key-generation protocol of Doerner et al. can be generalized to arbitrary thresholds, their signing scheme is in a few respects inherently limited to two parties. As with prior two-party schemes, it uses a Diffie-Hellman Key Exchange [DH76] to calculate the signature's *instance key* $R = k \cdot G$ (where G is the elliptic curve group generator), given a multiplicative sharing of k. With a threshold larger than two, the long-standing open problem of multiparty key exchange is implicated. A direct extension of the Diffie-Hellman method to t parties would require $t - 1$ rounds, and, though key exchange can be achieved in a sublinear number of rounds via indistinguishability obfuscation [BZ14] in the general case, or bilinear pairings [Jou04] when $t = 3$, neither of these methods results in a practically-efficient protocol with ECDSA-native assumptions. Additionally, the consistency check that ensures security against malicious adversaries is a decidedly two-party construction: it relies upon the asymmetrical roles of the parties, and integrates proof-wise with the aforementioned Diffie-Hellman Exchange. Furthermore, we note that the scheme of Doerner et al. realizes a nonstandard, two-party specific functionality. Though they prove in the Generic Group Model that this functionality confers no additional power to an adversary, it does allow one party to negligibly bias the distribution of the instance key.

In this work, we describe an extension of the protocols of Doerner et al. to arbitrary thresholds. We formally define a new multiparty functionality, replace their key exchange component with an alternative based on multiparty inverse sampling, develop a new consistency check, optimize the underlying primitives for the new setting and protocol structure, and prove our protocol secure in the Universal Composability (UC) paradigm [Can01]. We implement and benchmark our protocols, showing in particular that 256 LAN-connected parties can sign in about half of a second, and that 256 globally-distributed parties require about four seconds.

1.1 Our Techniques

Recall that an elliptic curve is defined by the tuple $(\mathbb{G}, G, q)$, where $\mathbb{G}$ is the group of order q of points on the curve, and G is the generator for that group. An ECDSA Signature on a message m under the secret key sk comprises a pair (sig, r_x) of integers in $\mathbb{Z}_q$ such that

$$\mathsf{sig} = \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$

where k is a uniform element from $\mathbb{Z}_q$ and r_x is the x-coordinate of the elliptic curve point $R = k \cdot G$. We frame our task as the construction of a multiparty computation at the end of which participating parties obtain additive shares of such a signature, having supplied secret shares of sk as input. We also require a protocol for generating shares of sk and for performing one-time initialization, which we refer to as the *setup* protocol.

Our setup protocol is a natural extension of Doerner et al. [DKLs18], requiring only minor changes to ensure security against a dishonest majority of participants. When it completes successfully, each of the n participating parties receives a point on a $(t-1)$-degree polynomial. The y-intercept of this polynomial is the secret key sk, as per Shamir's secret sharing scheme [Sha79]. This allows any group of t parties to obtain an additive sharing of sk using the appropriate Lagrange coefficients. This additive sharing is the input to our signing protocol. On the other hand, our signing protocol diverges from that of Doerner et al. It can be understood in terms of three logical phases.

1. *Multiplication and Inversion.* Once a group of parties $\mathbf{P}$ (where $|\mathbf{P}| = t$) have agreed to sign a message, they use a t-party inverse-sampling protocol to sample k. From this protocol they receive an additive sharing of k, an additive sharing of $1/k$, and the value $R = k \cdot G$. They then use a GMW-style [GMW87] multiplication protocol to compute an additive sharing of sk/k from their additive sharings of $1/k$ and sk.

2. *Consistency Check.* The parties use the public values $R = k \cdot G$ and $\mathsf{pk} = \mathsf{sk} \cdot G$ to verify that consistent and correct inputs were used in the previous phase. Each party broadcasts a set of values that sum to predictable targets if and only if all parties have used inputs for the GMW-style multiplier that are consistent with the outputs of the inverse-sampling protocol. This consistency check is similar in form and purpose to the consistency check employed by Doerner et al., but it operates in a broadcast fashion and enforces additional relationships.

3. *Signing.* Once the consistency of the previous phases has been checked, each party i in the set of participants $\mathbf{P}$ is convinced that it holds v_i, w_i, and R such that for some value k,

$$\sum_{i \in \mathbf{P}} v_i = \frac{1}{k} \qquad \text{and} \qquad \sum_{i \in \mathbf{P}} w_i = \frac{\mathsf{sk}}{k} \qquad \text{and} \qquad R = k \cdot G$$

The parties locally compute their shares of the signature

$$\mathsf{sig}_i := v_i \cdot H(m) + w_i \cdot r_x$$

and broadcast them. The signature is then reconstructed

$$\mathsf{sig} := \sum_{i \in \mathbf{P}} \mathsf{sig}_i$$

and verified using the standard verification algorithm.

Our signing protocol is therefore essentially composed of a maliciously secure t-party inverse-sampling protocol and a t-party multiplier, plus a check message to enforce consistency between the two. Both the t-party inverse-sampling protocol and the t-party multiplier are instantiated via a two-party multiplication protocol, with one instance of this protocol being run between each pair of parties. The asymptotic round count of the overall protocol is determined by the fact that data dependencies in the inverse-sampling process require these two-party multiplication protocols to be evaluated $\lceil \log(t) \rceil$ sequential groups.

Our two-party multiplier is based upon Oblivious Transfer (OT) and derived from the two-party multiplication protocol of Doerner et al. [DKLs18], who were inspired in turn by the semi-honest multiplication protocol of Gilboa [Gil99]. We improve upon the performance of their protocol in terms of both communication and computation. The protocol of Doerner et al. specifies that one of the two parties encodes its input using a high-entropy encoding scheme, and the length of this encoded input determines the number of OT instances required, which in turn strongly determines the performance of the multiplication protocol as a whole. On the other hand, our new protocol specifies that both parties choose random inputs, and later send correction messages to adjust their output values as necessary. Allowing only for encodings of random values simplifies the encoding scheme considerably and reduces the number of OT instances in proportion to the ECDSA security parameter, or about 40% in practice. This improvement comes at the cost of one additional round, but because the first two rounds are input independent and therefore parallelizable, our new multiplier actually reduces the overall round count of our ECDSA signing protocol relative to the multiplication protocol of Doerner et al.

1.2 Contributions

1. We present a t-of-n threshold ECDSA signing protocol that requires $\lceil \log(t) \rceil + 6$ rounds[1] and prove it UC-secure against a malicious adversary who statically corrupts $t - 1$ participants using only the Computational Diffie-Hellman Assumption in the Global Random Oracle Model. In addition we modify the setup protocol of Doerner et al. [DKLs18] and prove it secure under the same constraints.

[1] In an in-progress follow-up work, we reduce this to a constant by adapting the modular inversion protocol of Bar-Ilan and Beaver [BB89], at the cost of slightly larger constants.

2. We introduce a t-party inverse-sampling primitive, which may be of independent interest. This primitive requires $\lceil \log(t) \rceil + 6$ rounds, and we prove it statistically secure against a malicious adversary who statically corrupts $t - 1$ participants in the UC framework.

3. We improve upon the two-party multiplication protocol of Doerner et al., achieving a concrete performance gain of roughly 40%. In our protocol, a randomized Gilboa-style multiplier generates an unauthenticated multiplication triple, which is later adjusted at the cost of communicating a single field element for each party. Our protocol also supports batched multiplications, with a reduction in communication relative to simple repetition.

4. We provide an implementation of our protocol in the Rust language, and benchmark it on commodity server-class hardware in both the WAN and LAN settings, as well as on embedded devices. In the LAN setting, we evaluate our protocol with up to 256 parties. In the WAN setting, we evaluate with 256 parties spread across 16 datacenters. With respect to signing, our scheme outperforms all prior and concurrent work in the LAN setting by a factor of 10 or more, and it is competitive in the WAN setting in spite of its round count. With respect to setup, our scheme outperforms all prior and concurrent works by a factor of at least 100.

1.3 Contemporary Works

This work is contemporary to two other schemes for threshold ECDSA signing. The first, proposed by Lindell et al. [LNR18], derives from the earlier work of Gennaro et al. [GGN16], but mostly avoids the use of Paillier encryption, relying instead upon a novel technique which they refer two as "ElGamal in the Exponent". In their scheme, the parties compute their signature in a way that guarantees privacy but not correctness, while simultaneously computing the same signature in the elliptic curve group (i.e. "in the exponent") in a way that guarantees correctness but does not allow the signature to be reconstructed; the latter computation is used to verify the former. Lindell et al. make black-box use of a multiplication functionality, which they instantiate using either Paillier encryption or Oblivious Transfer, though they focus on the former case. In the end they prove their system simulation-secure under the Decisional Diffie-Hellman Assumption and (if Paillier is used to instantiate multiplication) the Decisional Composite Residuosity Assumption. They claim that if all component protocols are UC-secure, then their protocol achieves UC-security as well.

In the second contemporary scheme, from Gennaro and Goldfeder [GG18], the parties compute their signature in a way that does not guarantee correctness, and then use zero-knowledge techniques to verify that the signature is well-formed before revealing it by evaluating a randomized version of the verification equation in the elliptic curve group. Like Lindell et al., they use a Paillier-based multiplication protocol. They prove that their scheme fulfills a game-based security definition under the Decisional Diffie-Hellman Assumption, the Strong

RSA Assumption, and the assumption that ECDSA is a signature scheme. They also propose an optimized variant that requires an ad-hoc assumption over the Paillier cryptosystem, similar to the one introduced by Lindell [Lin17].

In this work, we take a substantially different approach from our contemporaries. Rather than invoking the ECDSA equations explicitly, we protect against malicious adversaries using a consistency check that verifies a select number of important relationships in the exponent, and rather than founding our protocol on Paillier-based multiparty multiplication, we use an inverse-sampling primitive based upon Oblivious Transfer. This approach allows us to achieve UC-security, a notion stronger than that achieved by Gennaro and Goldfeder, under only the Computational Diffie-Hellman Assumption, an assumption strictly weaker than the Decisional Diffie-Hellman Assumption required by both contemporary works. Unlike the others, we completely avoid the black-box use of non-interactive zero-knowledge techniques in our signing protocol, which carry a significant practical performance penalty if UC-security is to be achieved by an implementation. On the other hand, our approach incurs somewhat larger communication costs than the others, and requires a logarithmic number of rounds, whereas the other protocols have constant round counts. We discuss these issues further in Section 6, and in Section 7 we provide a comparison of concrete performance.

1.4 Organization

We establish the notation and building blocks for our protocols in Section 2. We describe our improved protocol for two-party multiplication in Section 3, which we use to construct t-party inverse sampling in Section 4. We specify our t-of-n threshold ECDSA protocol in Section 5. We analyze the cost of this protocol in Section 6 and provide details of our implementation and its performance in Section 7. Finally, we prove our protocols secure in the appendices.

2 Preliminaries and Definitions

2.1 Notation

Throughout this paper, we use $(\mathbb{G}, G, q)$ to represent the elliptic curve over which signatures are calculated, where $\mathbb{G}$ is the group of curve points, G the curve generator, and q the order of the curve. Curve points are represented in $|q| = \kappa$ bits, which is also the curve's security parameter, and we use s to represent the statistical security parameter. Curve points are denoted with capitalized variables and scalars with lower case. Vectors are given in bold and indexed by subscripts; thus $\mathbf{x}_i$ is the i^{th} element of the vector $\mathbf{x}$, which is distinct from the scalar variable x. We use $=$ for equality, $\coloneqq$ for assignment, $\leftarrow$ for sampling from a distribution, $\overset{c}{\equiv}$ for computational indistinguishability, and $\overset{s}{\equiv}$ for statistical indistinguishability. We make use of a Global Random Oracle [CJS14] $H^x(y) : \{0,1\}^* \to \mathbb{Z}_q^x$ with its output length varying according to the function's superscript; when the superscript is omitted it is assumed to be 1. We use $\mathcal{P}_i$

to denote the party with index i, and variables may often be subscripted with an index to indicate that they belong to a particular party. When arrays are owned by a party, the party index always comes before the array index. For convenience, when only two parties are present in a context, they are referred to as Alice and Bob.

In functionalities, we leave standard bookkeeping implicit. In particular, we assume that along with the other messages we specify, session IDs and party IDs are transmitted so that the functionality knows to which instance a message belongs and who is participating in that instance. We assume that the functionality aborts if a party tries to reuse a session ID, send messages out of order, etc. We use `slab-serif` to denote message tokens, which communicate the function of a message to its recipients. For simplicity, we omit from a functionality's specifier all parameters that we do not actively use. For example, many of our functionalities are parameterized by a group $\mathbb{G}$ of order q; we leave it implicit because in any instantiation all functionalities use the same group.

2.2 Digital Signatures

Definition 1 (Digital Signature Scheme [KL15])**.**
A *Digital Signature Scheme* is a tuple of probabilistic polynomial time algorithms, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ such that:

1. Given a security parameter κ, the Gen algorithm outputs a public key/secret key pair: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$
2. Given a secret key sk and a message m, the Sign algorithm outputs a signature σ: $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$
3. Given a message m, signature σ, and public key pk, the Verify algorithm outputs a bit b indicating whether the signature is valid: $b := \mathsf{Verify}_{\mathsf{pk}}(m, \sigma)$

A Digital Signature Scheme satisfies two properties:

1. (Correctness) With overwhelmingly high probability, all valid signatures must verify. Formally, over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and all messages m,
$$\Pr_{\mathsf{pk},\mathsf{sk},m}\left[\mathsf{Verify}_{\mathsf{pk}}(m, \mathsf{Sign}_{\mathsf{sk}}(m)) = 1\right] > 1 - \mathrm{negl}(\kappa)$$
2. (Existential Unforgeability) No adversary can forge a signature for any message with greater than negligible probability, even if that adversary has seen signatures for polynomially many messages of its choice. Formally, for all PPT adversaries $\mathcal{A}$ with access to the signing oracle $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$, where $\mathbf{Q}$ is the set of queries $\mathcal{A}$ asks the oracle,
$$\Pr_{\mathsf{pk},\mathsf{sk}}\left[\begin{array}{r}\mathsf{Verify}_{\mathsf{pk}}\,(m,\sigma) = 1 \wedge m \notin \mathbf{Q} : \\ (m,\sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}\,(\mathsf{pk})\end{array}\right] < \mathrm{negl}(\kappa)$$

2.3 ECDSA

ECDSA is parameterized by a group $\mathbb{G}$ of order q generated by a point G on an elliptic curve over the finite field $\mathbb{Z}_p$ of integers modulo a prime p. Assuming a curve has been fixed, the ECDSA algorithms are as follows [KL15]:

Algorithm 1. $\mathsf{Gen}(1^\kappa)$**:**

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. $\mathsf{Sign}(\mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$**:**

1. Uniformly choose an instance key $k \leftarrow \mathbb{Z}_q$.
2. Calculate $(r_x, r_y) = R := k \cdot G$.
3. Calculate
$$\mathsf{sig} := \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
4. Output $\sigma := (\mathsf{sig} \mod q, r_x \mod q)$.

Algorithm 3. $\mathsf{Verify}(\mathsf{pk} \in \mathbb{G}, m \in \{0,1\}^*, \sigma \in (\mathbb{Z}_q, \mathbb{Z}_q))$**:**

1. Parse σ as (sig, r_x).
2. Calculate
$$(r'_x, r'_y) = R' := \frac{H(m) \cdot G + r_x \cdot \mathsf{pk}}{\mathsf{sig}}$$
3. Output 1 if and only if $(r'_x \mod q) = (r_x \mod q)$.

2.4 Requisite Functionalities

In this section we introduce a small set of functionalities that we use as building blocks. We begin with a commitment functionality and a committed-zero-knowledge functionality. Informally, the commitment functionality $\mathcal{F}_{\mathsf{Com}}^n$ allows a party to send a commitment to a message to a group of parties, and later reveal the same message to these parties. The functionality $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$ allows a party to send a commitment to both an elliptic curve point and a proof of knowledge of its discrete logarithm to a group of parties, and later reveal both. Concretely, $\mathcal{F}_{\mathsf{Com}}^n$ can be instantiated via the folkloric hash-based construction, and $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$ via the Schnorr [Sch89] protocol made non-interactive using the Fiat-Shamir [FS86] or Fischlin [Fis05] transform, though only the latter achieves UC-security.

Functionality 1. Commitment ($\mathcal{F}_{\mathsf{Com}}^{n}$):

This functionality runs with a group of n parties, where one specific party $\mathcal{P}_i$ commits, and all other parties receive the commitment and committed value.

Commit: On receiving $(\texttt{commit}, \mathsf{id}^{\mathsf{com}}, x, \mathbf{I})$ from party $\mathcal{P}_i$ where $\mathbf{I} \subseteq [1, n]$, if $(\texttt{commit}, \mathsf{id}^{\mathsf{com}}, \cdot, \cdot)$ does not exist in memory, then store $(\texttt{commit}, \mathsf{id}^{\mathsf{com}}, x, \mathbf{I})$ in memory and send $(\texttt{committed}, \mathsf{id}^{\mathsf{com}}, i)$ to all parties $\mathcal{P}_j$ for $j \in \mathbf{I}$.

Decommit: On receiving $(\texttt{decommit}, \mathsf{id}^{\mathsf{com}})$ from $\mathcal{P}_i$, send $(\texttt{decommitted}, \mathsf{id}^{\mathsf{com}}, x)$ to every party $\mathcal{P}_j$ for $j \in \mathbf{I}$

Functionality 2. Committed ZKPoK for Discrete Log $\left(\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathrm{DL}}, n}\right)$:

This functionality is parameterized by the party count n and the elliptic curve $(\mathbb{G}, G, q)$. In each instance, one party $\mathcal{P}_i$ is the prover, and the others verify.

Commit Proof: On receiving $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}, x, X, \mathbf{I})$ from party $\mathcal{P}_i$ where $x \in \mathbb{Z}_q$ and $X \in \mathbb{G}$, if $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}, \cdot, \cdot, \cdot)$ does not exist in memory, then send $(\texttt{committed}, \mathsf{id}^{\mathsf{com\text{-}zk}}, i)$ to every party $\mathcal{P}_j$ for $j \in \mathbf{I}$ and store $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}, x, X, \mathbf{I})$ in memory.

Decommit Proof: On receiving $(\texttt{decom-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}})$ from party $\mathcal{P}_i$, if there exists in memory a record $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}, x, X, \mathbf{I})$, then:

1. If $X = x \cdot G$, send $(\texttt{accept}, \mathsf{id}^{\mathsf{com\text{-}zk}}, X)$ to every party $\mathcal{P}_j$ for $j \in \mathbf{I}$.
2. Otherwise send $(\texttt{fail}, \mathsf{id}^{\mathsf{com\text{-}zk}})$ to every $\mathcal{P}_j$ for $j \in \mathbf{I}$.

In addition, our multiplication protocols make use of Correlated Oblivious Transfer extensions [Bea96], which we model using the $\mathcal{F}_{\mathsf{COTe}}^{\eta}$ functionality of Doerner et al. [DKLs18], reproduced here for completeness. In short, $\mathcal{F}_{\mathsf{COTe}}^{\eta}$ interacts with two parties: A sender, who supplies a vector of correlations, and a receiver, who supplies vector of choice bits. For each vector element, the functionality returns to the sender a random pad, and to the receiver either the same random pad, or the same pad plus the sender's correlation. Concretely, we instantiate this functionality in the same manner as Doerner et al., using the OT-extension protocol of Keller et al. [KOS15], with Doerner et al.'s VSOT (a derivative of Simplest OT [CO15]) as the base OT.

Functionality 3. Correlated Oblivious Transfer Extensions ($\mathcal{F}_{\mathsf{COTe}}^{\eta}$):

This functionality is parameterized by the a batch size η and a set of groups $\{\mathbb{G}_i\}_{i \in [1,\eta]}$, one group for each element in a batch (though groups are not necessarily unique). It interacts with a sender S and a receiver R, who may run the Init phase once, and the Choice and Transfer phases many times.

Init: On receiving $(\texttt{init})$ from both parties, store $(\texttt{ready})$ in memory and send $(\texttt{init-complete})$ to the receiver.

Choice: On receiving $\left(\texttt{choose}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\beta}\right)$ from the receiver, if $\left(\texttt{choice}, \mathsf{id}^{\mathsf{ext}}, \cdot\right)$ with the same $\mathsf{id}^{\mathsf{ext}}$ does not exist in memory, and if $(\texttt{ready})$ does exist in memory, and if $\boldsymbol{\beta} \in \{0,1\}^{\eta}$, then send $\left(\texttt{chosen}, \mathsf{id}^{\mathsf{ext}}\right)$ to the sender and store $\left(\texttt{choice}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\beta}\right)$ in memory.

Transfer: On receiving $\left(\texttt{transfer}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\alpha}\right)$ from the sender, if a message of the form $\left(\texttt{choice}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\beta}\right)$ exists in memory with the same $\mathsf{id}^{\mathsf{ext}}$, and if $\left(\texttt{complete}, \mathsf{id}^{\mathsf{ext}}\right)$ does not exist in memory, and if $\boldsymbol{\alpha} \in \mathbb{G}_1 \times \ldots \times \mathbb{G}_\eta$, then:

1. Sample a vector of random pads $\boldsymbol{\omega}_{\mathsf{S}} \leftarrow \mathbb{G}_1 \times \ldots \times \mathbb{G}_\eta$
2. Send $\left(\texttt{pad}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}_{\mathsf{S}}\right)$ to the sender.
3. Compute $\boldsymbol{\omega}_{\mathsf{R}} \coloneqq \left\{\boldsymbol{\beta}_i \cdot \boldsymbol{\alpha}_i - \boldsymbol{\omega}_{\mathsf{S},i}\right\}_{i \in [1,\eta]}$.
4. Send $\left(\texttt{correlation}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}_{\mathsf{R}}\right)$ to the receiver.
5. Store $(\texttt{complete}, \mathsf{id}^{\mathsf{ext}})$ in memory.

3 Improved Two-party Multiplication

Like Doerner et al. [DKLs18], we build our signing protocol atop two-party multiplication. They introduced a multiplication protocol optimized for the single-use computation setting (in which a small number of multiplications are computed by exactly two parties with no preprocessing), and we will now introduce a variant of their protocol that is optimized for scenarios in which multiple overlapping pairs of parties compose their multiplications with one another. As a result we require a new functionality, conceptually similar to the functionality realized by Beaver triples [Bea91]. In our case Alice and Bob learn a and b respectively, plus one share each of z such that $a \cdot b = z$, whereas in the case of Beaver triples, Alice and Bob learn one share each of a, b, and z.

Our functionality involves three main phases. Following the one-time initialization phase, there is a preprocessing phase in which the parties must each send a message to the functionality in a specific order. Following this, they can supply their inputs (either party going first), and as each party's input is supplied, the opposite party's output is delivered. One party is also given the ability to define their own output by rushing in the last phase; we will discuss this later in conjunction with the protocol that realizes our functionality. When our functionality is composed, multiple instances can preprocess concurrently, and then inputs can be supplied as data dependencies require. This corresponds to a savings in rounds in our higher-level protocols.

In addition, we add to both our protocol and our functionality the ability to batch multiplications together, and we make a simplification relative to Doerner et al.: whereas their two-party multiplication functionality allows an adversary to inject additive error into the output, ours does not. Because both functionalities output unauthenticated shares, an offset can always be induced.

Functionality 4. Two-party Multiplication ($\mathcal{F}^{\ell}_{\mathsf{2PMul}}$):

This functionality is parameterized by the batch size ℓ and the group $\mathbb{Z}_q$ over which multiplication is to be performed. It interacts with two parties, Alice and Bob, who may run the Init phase once, the remaining phases repeatedly.

Init: Wait for message (`init`) from Alice and Bob. Store (`init-complete`) in memory and send (`init-complete`) to Bob.

Bob's Preprocessing: On receiving $(\mathtt{preprocess}, \mathsf{id}^{\mathsf{mul}})$ from Bob, if a record $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ with the same $\mathsf{id}^{\mathsf{mul}}$ does not exist in memory, but the record (`init-complete`) does exist in memory, then store $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ in memory, and send $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ to Alice.

Alice's Preprocessing: On receiving $(\mathtt{preprocess}, \mathsf{id}^{\mathsf{mul}})$ from Alice, if there exists a record $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}}, \cdot)$ in memory with the same $\mathsf{id}^{\mathsf{mul}}$, and if $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ does not exist in memory, then store $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ in memory, and send $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ to Bob.

Alice's Input: On receiving $(\mathtt{input}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a})$ from Alice, if $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ exists in memory with the same $\mathsf{id}^{\mathsf{mul}}$, and if $(\mathtt{alice\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \cdot, \cdot)$ and $(\mathtt{bob\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \cdot, \cdot)$ do not exist in memory, and if $\mathbf{a} \in \mathbb{Z}_q$ then

1. Sample $\mathbf{z}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\ell}$
2. Send $(\mathtt{output}, \mathsf{id}^{\mathsf{mul}}, \mathbf{z}_{\mathsf{B}})$ to Bob.
3. Store $(\mathtt{alice\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a}, \mathbf{z}_{\mathsf{B}})$ in memory.

Bob's Input: On receiving $(\mathtt{input}, \mathsf{id}^{\mathsf{mul}}, \mathbf{b})$ from Bob, if there exists a record $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ in memory with the same $\mathsf{id}^{\mathsf{mul}}$, and if no record of the form $(\mathtt{bob\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \cdot, \cdot)$ exists in memory, and if $\mathbf{b} \in \mathbb{Z}_q$ then

1. If $(\mathtt{alice\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a}, \mathbf{z}_{\mathsf{B}})$ exists in memory, then compute
$$\mathbf{z}_{\mathsf{A}} \coloneqq \left\{\mathbf{a}_i \cdot \mathbf{b}_i - \mathbf{z}_{\mathsf{B},i}\right\}_{i \in [1,\ell]}$$
and send $(\mathtt{output}, \mathsf{id}^{\mathsf{mul}}, \mathbf{z}_{\mathsf{A}})$ to Alice.
2. Otherwise send $(\mathtt{bob\text{-}complete}, \mathsf{id}^{\mathsf{mul}})$ to Alice.
3. Store $(\mathtt{bob\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \mathbf{b})$ in memory.

Rushing Alice: On receiving $(\mathtt{rush}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a}, \mathbf{z}_{\mathsf{A}})$ from Alice, if records $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ and $(\mathtt{bob\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \mathbf{b})$ exist in memory, but $(\mathtt{alice\text{-}complete}, \mathsf{id}^{\mathsf{mul}}, \cdot, \cdot)$ does not, and if $\mathbf{a} \in \mathbb{Z}_q$ and $\mathbf{z}_{\mathsf{A}} \in \mathbb{Z}_q$ then

1. Compute
$$\mathbf{z}_{\mathsf{B}} \coloneqq \left\{\mathbf{a}_i \cdot \mathbf{b}_i - \mathbf{z}_{\mathsf{A},i}\right\}_{i \in [1,\ell]}$$

2. Send $(\texttt{output}, \mathsf{id}^{\mathsf{mul}}, \mathbf{z}_{\mathsf{B}})$ to Bob.

3. Store $(\texttt{alice-complete}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a}, \mathbf{z}_{\mathsf{B}})$ in memory.

The protocol with which we instantiate the foregoing functionality is based upon Correlated Oblivious Transfer Extensions. Alice and Bob use oblivious transfer to perform a *randomized* multiplication, in a similar style to the non-randomized multiplication of Doerner et al., and they adjust their output shares after the fact when their inputs become known. The randomized multiplication requires two rounds on its own, and the adjustment step a third, as compared to the two-round multiplier of Doerner et al., but in our case it is possible for many multipliers with data dependencies to evaluate their randomized multiplications concurrently, reducing the round count overall. This corresponds to parallelizing the preprocessing phases of multiple $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ instances.

Protocol 1. Two-party Multiplication $\left(\pi^{\ell}_{\mathsf{2PMul}}\right)$:

This protocol is parameterized by the batch size ℓ, the statistical security parameter s, and the group $\mathbb{Z}_q$ over which multiplication is to be performed. Let $\kappa = |q|$ and for convenience let $\eta = \xi \cdot \ell$ where $\xi = \kappa + 2s$ is the number of random choice bits per element in a batch. This protocol makes use of a public *gadget vector* $\mathbf{g} \leftarrow \mathbb{Z}_q^{\eta}$, and it invokes the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ functionality and the random oracle H. Alice and Bob supply a vectors input integers $\mathbf{a} \in \mathbb{Z}_q^{\ell}$ and $\mathbf{b} \in \mathbb{Z}_q^{\ell}$ respectively. They receive as output vectors of integers $\mathbf{z}_{\mathsf{A}} \in \mathbb{Z}_q^{\ell}$ and $\mathbf{z}_{\mathsf{B}} \in \mathbb{Z}_q^{\ell}$.

Init: Alice and Bob transmit $(\texttt{init})$ to $\mathcal{F}^{\eta}_{\mathsf{COTe}}$.

Randomized Multiplication:

1. Bob samples a set of OT choice bits and calculates his pads $\tilde{\mathbf{b}}$

$$\boldsymbol{\beta} \leftarrow \{0,1\}^{\eta} \qquad \text{and} \qquad \tilde{\mathbf{b}} \coloneqq \left\{ \left\langle \mathbf{g}, \left\{ \boldsymbol{\beta}_{i \cdot \xi + j} \right\}_{j \in [1,\xi]} \right\rangle \right\}_{i \in [0,\ell)}$$

2. Alice samples a set of pads $\tilde{\mathbf{a}} \leftarrow \mathbb{Z}_q^{\ell}$ and check values $\hat{\mathbf{a}} \leftarrow \mathbb{Z}_q^{\ell}$ and sets her OT correlation $\boldsymbol{\alpha} \in \mathbb{Z}_q^{\eta}$ as

$$\boldsymbol{\alpha} \coloneqq \{\tilde{\mathbf{a}}_1 \| \hat{\mathbf{a}}_1\}_{j \in [1,\xi]} \parallel \ldots \parallel \{\tilde{\mathbf{a}}_{\ell} \| \hat{\mathbf{a}}_{\ell}\}_{j \in [1,\xi]}$$

3. Alice and Bob access the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ functionality, supplying $\eta = \xi \cdot \ell$ as the OT-extension batch size. Alice plays the sender, supplying $\boldsymbol{\alpha}$ as her input, and Bob, the receiver, supplies $\boldsymbol{\beta}$. They receive as outputs, respectively, the arrays $\boldsymbol{\omega}_{\mathsf{A}} \in \mathbb{Z}_q^{\eta}$ and $\boldsymbol{\omega}_{\mathsf{B}} \in \mathbb{Z}_q^{\eta}$, which they interpret as

$$\left\{ \tilde{\mathbf{z}}_{\mathsf{A},j} \middle\| \hat{\mathbf{z}}_{\mathsf{A},j} \right\}_{j \in [1,\eta]} = \boldsymbol{\omega}_{\mathsf{A}} \qquad \text{and} \qquad \left\{ \tilde{\mathbf{z}}_{\mathsf{B},j} \middle\| \hat{\mathbf{z}}_{\mathsf{B},j} \right\}_{j \in [1,\eta]} = \boldsymbol{\omega}_{\mathsf{B}}$$

That is, $\tilde{\mathbf{z}}_{\mathsf{A}}$ is a vector wherein each element contains the first half of the corresponding element in Alice's output from $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, and $\hat{\mathbf{z}}_{\mathsf{A}}$ is a vector wherein each element contains the second half. $\tilde{\mathbf{z}}_{\mathsf{B}}$ and $\hat{\mathbf{z}}_{\mathsf{B}}$ play identical roles for Bob. The steps in the protocol up to this point correspond to Bob's preprocessing phase in $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

4. Alice and Bob generate 2ℓ shared, random values by calling the random oracle. As input they use the shared components of the transcript of the protocol that implements $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, so that that these values have a dependency on the completion of Step 3. In our proofs, we abstract this step as a coin tossing protocol.

$$\tilde{\boldsymbol{\chi}} \leftarrow H^{\ell}(1 \| \mathsf{transcript}) \qquad \text{and} \qquad \hat{\boldsymbol{\chi}} \leftarrow H^{\ell}(2 \| \mathsf{transcript})$$

5. Alice computes

$$\mathbf{r} \coloneqq \left\{ \sum_{i \in [0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \right\}_{j \in [1,\xi]}$$
$$\mathbf{u} \coloneqq \{\tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{a}}_i + \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{a}}_i\}_{i \in [1,\ell]}$$

and sends $\mathbf{r}$ and $\mathbf{u}$ to Bob.

6. Bob aborts if

$$\bigvee_{j \in [1,\xi]} \left(\begin{array}{r} \mathbf{r}_j + \sum_{i \in [0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\mathbf{z}}_{\mathsf{B},i \cdot \xi + j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\mathbf{z}}_{\mathsf{B},i \cdot \xi + j} \\ \neq \sum_{i \in [0,\ell)} \boldsymbol{\beta}_{i \cdot \xi + j} \cdot \mathbf{u}_{i+1} \end{array} \right)$$

Note that steps 4, 5, and 6 correspond to Alice's preprocessing in $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

Input and Adjustment:

7. Alice and Bob respectively compute

$$\boldsymbol{\gamma}_{\mathsf{A}} \coloneqq \{\mathbf{a}_i - \tilde{\mathbf{a}}_i\}_{i \in [1,\ell]} \qquad \text{and} \qquad \boldsymbol{\gamma}_{\mathsf{B}} \coloneqq \left\{\mathbf{b}_i - \tilde{\mathbf{b}}_i\right\}_{i \in [1,\ell]}$$

and send these values to one another.

8. Alice and Bob compute their output shares

$$\mathbf{z}_{\mathsf{A}} \coloneqq \left\{ \mathbf{a}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{B},i+1} + \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \right\}_{i \in [0,\ell)}$$

$$\mathbf{z}_{\mathsf{B}} := \left\{ \tilde{\mathbf{b}}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{A},i+1} + \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} \right\}_{i \in [0,\ell)}$$

Note that Steps 7 and 8 correspond to the final three phases in $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

Theorem 3.1. *When $\ell \leq c \cdot \log(\kappa)$ for some constant c, the protocol $\pi^{\ell}_{\mathsf{2PMul}}$ UC-realizes the functionality $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ for a κ-bit field $\mathbb{Z}_q$ in the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$-hybrid Global Random Oracle Model, in the presence of a malicious adversary statically corrupting either party.*

Proof. A proof of this theorem is given in Appendix B. □

Rushing Adversaries. In both $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and $\pi^{\ell}_{\mathsf{2PMul}}$ we specify that during the adjustment process, either Alice or Bob may adjust their value first. This ensures that both adjustments can occur in a single round, without assuming simultaneous message transmission: in the real world, one party will likely transmit slightly before the other, but neither party will know the transmission order until after both messages are sent. Due to the asymmetry in the equations that the parties use to calculate their output shares in Step 8 of $\pi^{\ell}_{\mathsf{2PMul}}$, this pseudo-simultaneous transmission opens up an opportunity for a rushing adversary to deprive the simulator of information that it requires to produce $\boldsymbol{\gamma}_{\mathsf{B}}$, which necessitates the addition of the aforementioned Rushing Alice phase in $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

Consider a similar functionality that lacked the final phase (always using the Alice's Input phase instead), and imagine the procedure of the simulator $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ that interacts with Alice and plays the role of the ideal adversary for $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. If Bob adjusts his input first, then $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ will communicate Alice's output $\mathbf{z}_{\mathsf{A}}$ to $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$. $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ must then calculate an adjustment value $\boldsymbol{\gamma}_{\mathsf{B}}$ that causes the output in her view to equal the value $\mathbf{z}_{\mathsf{A}}$ returned by $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. $\boldsymbol{\gamma}_{\mathsf{B}}$ must satisfy

$$\boldsymbol{\gamma}_{\mathsf{B}} = \left\{ \frac{\mathbf{z}_{\mathsf{A},i+1} - \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j}}{\mathbf{a}_{i+1}} \right\}_{i \in [0,\ell)}$$

At this point, $\mathbf{z}_{\mathsf{A}}$ and $\tilde{\mathbf{z}}_{\mathsf{A}}$ should be known to the simulator, but since Alice has not yet transmitted her adjustment message, the simulator should not know $\mathbf{a}$, and consequently the correct value of $\boldsymbol{\gamma}_{\mathsf{B}}$ cannot be calculated. We remedy this by compelling Alice to determine her own output $\mathbf{z}_{\mathsf{A}}$ in the ideal world if and only if she performs her adjustment second. Consequently $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ can choose $\boldsymbol{\gamma}_{\mathsf{B}}$ uniformly, and Alice's subsequent adjustment message $\boldsymbol{\gamma}_{\mathsf{A}}$ fixes her output $\mathbf{z}_{\mathsf{A}}$ and thereby allows her input to be extracted via the above equation. Note that when simulating against Bob, and equivalent problem does not occur, since the equation Bob uses to adjust his output value does not involve his input value. We formalize the intuition presented here in Appendix B.

Round Count. As we have mentioned, our multiplication protocol $\pi^{\ell}_{\mathsf{2PMul}}$ requires an additional round relative to the protocol of Doerner et al. This

third round is necessitated by the proof of security and not the protocol per se: the adjustment messages γ_{A} and γ_{B} have no data dependency upon the random multiplication that precedes them, but an adversary with knowledge of their counterparty's input could potentially use that knowledge in combination with the adjustment messages to compromise the random multiplication, were the adjustment messages sent before the random multiplication is complete. In Section 5, we present a context-dependent two-round optimization.

Cost Comparison to Prior Work. Our multiplication protocol incurs a cost of $\ell \cdot (\kappa + 2s)$ OT invocations for a batched input of size ℓ. The encoding scheme used by Doerner et al.'s multiplication protocol specifies codewords of size $2\kappa + 2s$, which implies a cost of $2\kappa + 2s$ OT instances per input. In practice, it is reasonable to choose $\kappa = 256$ and $s = 80$, under which parameterization our protocol yields a savings of about 40% in terms of OT instances.

4 Sampling Modular Inverses

In this section, we present a t-party modular inversion protocol based upon the folkloric technique of t-party multiplication via composition of $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. The ability of a malicious party playing the role of Alice to define its own output by rushing while interacting with $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ implies that an adversary that can do the same in the t-party setting, so long as at least one corrupted party plays the role of Alice. For simplicity, our functionality will assume that this is always the case and unconditionally allow corrupted parties to define their own output. We note that the simulator, which we will describe in Appendix C, cannot extract outputs from corrupted parties individually, but must instead extract the sum of all corrupted parties' outputs. Consequently, we specify that a single ideal adversary (the simulator $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$) interacts with the functionality in the corrupted parties' stead. In both the functionality and the protocol, a group of n parties run the setup phase, and size-t subgroups of these parties may then sample inverses.

Functionality 5. t-party Modular Inverse Sampling $\left(\mathcal{F}^{n,t}_{\mathsf{Inv}}\right)$:

This functionality is parameterized by the party count n, the threshold size t, and the elliptic curve group $(\mathbb{G}, G, q)$. The Init phase runs once with a group of n parties, and the Inversion phase may be run many times among any (varying) subgroup of parties indexed by $\mathbf{P} \subseteq [1, n]$ such that $|\mathbf{P}| = t$. An ideal adversary, denoted $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$, statically corrupts the parties indexed by the set $\mathbf{P}^* \subset [1, n]$ such that $|\mathbf{P}^*| < t$. Inputs from corrupt parties are provided directly to the functionality by $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$ as a single, combined value.

Init: Wait for message (`init`) from the honest parties $\{\mathcal{P}_i\}_{i\in[1,n]\setminus\mathbf{P}^*}$ and from $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$. Store (`init-complete`) in memory and send (`init-complete`) to $\{\mathcal{P}_i\}_{i\in[1,n]\setminus\mathbf{P}^*}$ and to $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$.

Sampling: Receive $(\mathtt{inv}, \mathsf{id}^{\mathsf{inv}}, \mathbf{P})$ from each party $\mathcal{P}_i$ for $i \in \mathbf{P} \setminus \mathbf{P}^*$, and receive $(\mathtt{inv}, \mathsf{id}^{\mathsf{inv}}, \mathbf{P}, u_*)$ from $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$. If (`init-complete`) exists in memory but

$(\texttt{shares}, \mathsf{id}^{\mathsf{inv}}, \cdot, \cdot)$ does not, and all parties agree to the same set $\mathbf{P}$, then sample

$$\{u_i\}_{i \in \mathbf{P} \setminus \mathbf{P}^*} \leftarrow \mathbb{Z}_q^{|\mathbf{P} \setminus \mathbf{P}^*|}$$

uniformly, compute

$$R := u_* \cdot G + \sum_{i \in \mathbf{P} \setminus \mathbf{P}^*} u_i \cdot G$$

and store $(\texttt{shares}, \mathsf{id}^{\mathsf{inv}}, \{u_i\}_{i \in \mathbf{P} \setminus \mathbf{P}^*}, u_*)$ in memory. Send $(\texttt{output}, \mathsf{id}^{\mathsf{inv}}, u_i)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P} \setminus \mathbf{P}^*$ and $(\texttt{exp-output}, \mathsf{id}^{\mathsf{inv}}, R)$ to $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, and upon receiving $(\texttt{exp-release}, \mathsf{id}^{\mathsf{inv}})$ from $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, send $(\texttt{exp-output}, \mathsf{id}^{\mathsf{inv}}, R)$ to $\mathcal{P}_i$ for $i \in \mathbf{P} \setminus \mathbf{P}^*$.

Inversion: Wait for either $(\texttt{inv-release}, \mathsf{id}^{\mathsf{inv}})$ or $(\texttt{inv-zero}, \mathsf{id}^{\mathsf{inv}})$ from $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$. If $(\texttt{inv-release}, \mathsf{id}^{\mathsf{inv}})$ is received, then sample $v_* \leftarrow \mathbb{Z}_q$, and if $(\texttt{inv-zero}, \mathsf{id}^{\mathsf{inv}})$ is received, then set $v_* := 0$. Regardless of which message is received, if $(\texttt{shares}, \mathsf{id}^{\mathsf{inv}}, \{u_i\}_{i \in \mathbf{P} \setminus \mathbf{P}^*}, u_*)$ exists in memory but $(\texttt{complete}, \mathsf{id}^{\mathsf{inv}})$ does not exist in memory, then sample

$$\{v_i\}_{i \in \mathbf{P} \setminus \mathbf{P}^*} \leftarrow \mathbb{Z}_q^{|\mathbf{P} \setminus \mathbf{P}^*|}$$

uniformly subject to

$$v_* + \sum_{i \in \mathbf{P} \setminus \mathbf{P}^*} v_i = \frac{1}{u_* + \sum_{i \in \mathbf{P} \setminus \mathbf{P}^*} u_i}$$

and send $(\texttt{inv-output}, \mathsf{id}^{\mathsf{inv}}, v_*)$ to $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$. Send $(\texttt{inv-output}, \mathsf{id}^{\mathsf{inv}}, v_i)$ as private, adversarially-delayed output to $\mathcal{P}_i$ for $i \in \mathbf{P} \setminus \mathbf{P}^*$, and store $(\texttt{complete}, \mathsf{id}^{\mathsf{inv}})$.

Our inverse-sampling protocol is based upon the folkloric technique of n-party multiplication. The parties sample multiplicative shares of k and a pad ϕ, and then locally compute multiplicative shares of ϕ/k. They use two n-party multipliers to convert their multiplicative shares of k and ϕ/k into additive shares, and then use a consistency check to verify that the two sets of shares are related in the correct way, after which they remove the pad.

Protocol 2. t-party Modular Inverse Sampling $\left(\pi_{\mathsf{Inv}}^{n,t}\right)$:

This protocol is parameterized by the party count n, the threshold size t, the statistical security parameter s, and elliptic curve group $(\mathbb{G}, G, q)$. It invokes the functionalities $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ and $\mathcal{F}_{\mathsf{Com}}^{n}$. The Init phase is run once with a group of n parties, and the subsequent phases can be run repeatedly by a varying subset of parties $\mathbf{P} \subseteq [1, n]$ such that $|\mathbf{P}| = t$. During each execution, every party $\mathcal{P}_i$ receives a pair of outputs $u_i, v_i \in \mathbb{Z}_q$ and they all receive a single value $R \in \mathbb{G}$.

Init: Each pair of parties $\mathcal{P}_i, \mathcal{P}_j$ for $i, j \in [1, n]$ such that $i < j$ initialize their multiplication oracle by sending $(\texttt{init})$ to their shared $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ instance.

Inverse Sampling:

1. Each party $\mathcal{P}_i$ for $i \in \mathbf{P}$ samples $k \leftarrow \mathbb{Z}_q$ and a pad $\phi_i \leftarrow \mathbb{Z}_q$ and sets

$\boldsymbol{\psi}_i^1 := \{k_i, \phi_i / k_i\}$.

2. Each party $\mathcal{P}_i$ commits to the pad by sending $(\texttt{commit}, \mathsf{id}_{i,1}^{\mathsf{com}}, \phi_i, \mathbf{P})$ to $\mathcal{F}_{\mathsf{Com}}^n$ using a fresh value for $\mathsf{id}_{i,1}^{\mathsf{com}}$. All other parties are notified of $\mathcal{P}_i$'s commitment.

3. For each pair of parties $\mathcal{P}_i, \mathcal{P}_j$ such that $i < j$:

   (a) $\mathcal{P}_j$, acting as Bob, sends $(\texttt{preprocess}, \mathsf{id}_{i,j}^{\mathsf{mul}})$ to $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, where $\mathsf{id}_{i,j}^{\mathsf{mul}}$ is a unique, agreed upon index, and $\ell = 2$.

   (b) On receiving $(\texttt{bob-ready}, \mathsf{id}_{i,j}^{\mathsf{mul}})$ from $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, $\mathcal{P}_i$, as Alice, sends $(\texttt{preprocess}, \mathsf{id}_{i,j}^{\mathsf{mul}})$ to $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$.

   (c) $\mathcal{P}_j$ receives $(\texttt{alice-ready}, \mathsf{id}_{i,j}^{\mathsf{mul}})$ from $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$.

4. For $\rho \in [1, \log_2(t)]$:

   (a) For each pair of parties $\mathcal{P}_i, \mathcal{P}_j$ in each contiguous non-overlapping subgroup of 2^ρ parties from $\mathbf{P}$, if $\mathcal{P}_i$ and $\mathcal{P}_j$ have not previously interacted during the course of this invocation of $\pi_{\mathsf{Inv}}^{n,t}$, then they send $(\texttt{input}, \mathsf{id}_{i,j}^{\mathsf{mul}}, \boldsymbol{\psi}_i^\rho)$ and $(\texttt{input}, \mathsf{id}_{i,j}^{\mathsf{mul}}, \boldsymbol{\psi}_j^\rho)$ to $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, respectively, and receive $(\texttt{output}, \mathsf{id}_{i,j}^{\mathsf{mul}}, \boldsymbol{\zeta}_i^{\rho,j})$ and $(\texttt{output}, \mathsf{id}_{i,j}^{\mathsf{mul}}, \boldsymbol{\zeta}_j^{\rho,i})$. If the party playing the role of Alice goes second, then it samples a random output and uses the Rushing Alice phase of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$.

   (b) Each party $\mathcal{P}_i$ privately computes $\boldsymbol{\psi}_i^{\rho+1}$ to be the element-wise sum of its output shares for round ρ:

$$\boldsymbol{\psi}_i^{\rho+1} := \left\{ \sum_{j \in \mathbf{P}^{\rho,i}} \boldsymbol{\zeta}_{i,l}^{\rho,j} \right\}_{l \in [1,2]}$$

   where $\mathbf{P}^{\rho,i} \subset \mathbf{P}$ is the subgroup with whom $\mathcal{P}_i$ interacted in round ρ such that $|\mathbf{P}^{\rho,i}| = 2^{\rho-1}$.

5. Each party $\mathcal{P}_i$ sets $\{u_i, \tilde{v}_i\} = \boldsymbol{\psi}_i^{\log_2(t)+1}$. Note that

$$\sum_{i \in \mathbf{P}} u_i = \prod_{i \in \mathbf{P}} k_i \qquad \text{and} \qquad \sum_{i \in \mathbf{P}} \tilde{v}_i = \prod_{i \in \mathbf{P}} \frac{\phi_i}{k_i}$$

Consistency Check:

6. Each party $\mathcal{P}_i$ computes $R_i := u_i \cdot G$ and commits to it by sending $(\mathtt{commit}, \mathsf{id}^{\mathsf{com}}_{i,2}, R_i, \mathbf{P})$ to $\mathcal{F}^n_{\mathsf{Com}}$, using a fresh value for $\mathsf{id}^{\mathsf{com}}_{i,2}$.

7. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ sends $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{i,2})$ to $\mathcal{F}^n_{\mathsf{Com}}$ and collects $\{R_j\}_{j \in \mathbf{P} \setminus \{i\}}$ as the other parties do the same. Then, each party computes

$$R := \sum_{j \in \mathbf{P}} R_j$$

8. Each party $\mathcal{P}_i$ calculates $\Gamma^1_i := \tilde{v}_i \cdot R$ and commits to it by sending $(\mathtt{commit}, \mathsf{id}^{\mathsf{com}}_{i,3}, \Gamma^1_i, \mathbf{P})$ to $\mathcal{F}^n_{\mathsf{Com}}$, using a fresh value for $\mathsf{id}^{\mathsf{com}}_{i,3}$.

9. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ for $i \in \mathbf{P}$ sends $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{i,1})$ and $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{i,3})$ to $\mathcal{F}^n_{\mathsf{Com}}$ and collects $\{(\phi_j, \Gamma^1_j)\}_{j \in \mathbf{P} \setminus \{i\}}$ as the other parties do the same.

10. Each party $\mathcal{P}_i$ computes

$$\phi := \prod_{j \in \mathbf{P}} \phi_j$$

and aborts if the pad is equal to zero, or if

$$\sum_{j \in \mathbf{P}} \Gamma^1_j \neq \phi \cdot G$$

11. Each party $\mathcal{P}_i$ computes $v_i := \tilde{v}_i / \phi$ and takes (u_i, v_i, R) as its output.

Theorem 4.1. *The protocol $\pi^{n,t}_{\mathsf{Inv}}$ UC-realizes the functionality $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ for a κ-bit elliptic curve group $(\mathbb{G}, G, q)$ in the $(\mathcal{F}^{\ell}_{\mathsf{2PMul}}, \mathcal{F}^n_{\mathsf{Com}})$-hybrid model, in the presence of a computationally unbound malicious adversary statically corrupting up to $t-1$ parties.*

Proof. A proof of this theorem is given in Appendix C. □

Round Count. The protocol $\pi^{n,t}_{\mathsf{Inv}}$ requires each party to engage in t instances of the $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ functionality. The preprocessing phases of these instances are evaluated in parallel, but due to data dependencies, the input-adjustment phases must be evaluated in $\lceil \log(t) \rceil$ sequential groups. The subsequent consistency check requires four rounds. When $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ is realized by $\pi^{\ell}_{\mathsf{2PMul}}$, preprocessing requires two rounds, and we can perform the commitment in Step 2 simultaneously with the first round of preprocessing. Thus $\pi^{n,t}_{\mathsf{Inv}}$ requires $\lceil \log(t) \rceil + 6$ rounds in the general case. In Section 5, we present a context-dependent optimization that reduces the round count to $\lceil \log(t) \rceil + 5$.

5 Threshold ECDSA

In this section, we describe our threshold ECDSA functionality, followed by setup and signing protocols that jointly realize it. We have made no attempt to formulate a general signature functionality, but instead have modeled ECDSA in the threshold setting directly, much as previous works [BGG17, Lin17, DKLs18] have done. We note that ECDSA makes use of a hash function, and that the standard specifies this function to be SHA-256. Thus, when the following functionality makes use of the function H, it refers not to the Random Oracle but to the concrete SHA-256 function. We note that while the two-party ECDSA functionality of Doerner et al. allowed malicious parties the ability to bias the instance key, our functionality does not.

Functionality 6. t-of-n ECDSA $\left(\mathcal{F}_{\mathsf{ECDSA}}^{n,t}\right)$:

This functionality is parameterized by the party count n, the threshold t, the elliptic curve $(\mathbb{G}, G, q)$, and a hash function H. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [1, n]$ such that $|\mathbf{P}| = t$.

Setup: On receiving (`init`) from all parties,

1. Sample and store the joint secret key, $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2. Compute and store the joint public key, $\mathsf{pk} := \mathsf{sk} \cdot G$.
3. Send (`public-key`, pk) to all parties.
4. Store (`ready`) in memory.

Signing: On receiving (`sign`, $\mathsf{id}^{\mathsf{sig}}$, $\mathbf{P}$, m) from all parties indexed by $\mathbf{P}$, if (`ready`) exists in memory but (`complete`, $\mathsf{id}^{\mathsf{sig}}$) does not exist in memory, and if all parties agree to the same set $\mathbf{P}$, then

1. Sample $k \leftarrow \mathbb{Z}_q$ and store it as the instance key.
2. Wait for (`get-instance-key`, $\mathsf{id}^{\mathsf{sig}}$) from all parties indexed by $\mathbf{P}$.
3. Compute $R := k \cdot G$ and send (`instance-key`, $\mathsf{id}^{\mathsf{sig}}$, R) to all parties indexed by $\mathbf{P}$. Let $(r_x, r_y) = R$.
4. Wait for (`proceed`, $\mathsf{id}^{\mathsf{sig}}$) from all parties indexed by $\mathbf{P}$.
5. Compute
$$\mathsf{sig} := \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
6. Collect the signature, $\sigma := (\mathsf{sig} \mod q, r_x \mod q)$.

7. Send $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \sigma)$ to all parties indexed by $\mathbf{P}$ as adversarially-delayed private output.

8. Store $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ in memory.

Our setup protocol is derived from the 2-of-n setup protocol of Doerner et al. [DKLs18]. Like their scheme, it uses simple techniques to produce and verify an n-party Shamir secret sharing [Sha79] of a joint secret key sk, from which any t parties can derive a t-party additive sharing of sk with no further interaction; unlike their scheme, we use a proof of knowledge to ensure security against a dishonest majority. This is the only use of zero-knowledge techniques in this work. Their protocol samples the public/private key pair as an n-party additive sharing, and then converts it to a Shamir sharing; we make a small improvement by sampling the Shamir sharing directly. Specifically, each party locally samples a random polynomial of degree-$(t-1)$ and distributes points at predetermined locations on this polynomial to the other parties. The parties sum the points they receive to construct a Shamir sharing of a single degree-$(t-1)$ polynomial. The parties then multiply their points on the shared polynomial by the elliptic curve generator G, broadcast the result, and verify that all subsets of their shares represent the same polynomial by homomorphically evaluating the polynomial in the curve group. For degree-2 polynomials, Doerner et al. required a number of evaluations quadratic in n, whereas we require only a linear number regardless of the polynomial degree. Since the homomorphic evaluation of the polynomial is equal to pk, an adversary can learn nothing more from the protocol than could be learned from any protocol that realizes the same functionality.

Protocol 3. t-of-n ECDSA Setup $\left(\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}\right)$:

This protocol is parameterized by the party count n, the threshold size t, and the elliptic curve $(\mathbb{G}, G, q)$. It invokes the $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, $\mathcal{F}^{n,t}_{\mathsf{Inv}}$, and $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$ functionalities. It takes no input and yields to each party $\mathcal{P}_i$ a point $p(i)$ on the polynomial p, and the joint public key pk.

Public Key Generation:

1. Each party $\mathcal{P}_i$ samples a random degree polynomial p_i of degree $t-1$.

2. For all pairs of parties $\mathcal{P}_i$ and $\mathcal{P}_j$, $\mathcal{P}_i$ sends $p_i(j)$ to $\mathcal{P}_j$ and receives $p_j(i)$ in return.

3. Each party $\mathcal{P}_i$ computes its point

$$p(i) \coloneqq \sum_{j \in [1,n]} p_j(i)$$

4. Each $\mathcal{P}_i$ computes $T_i \coloneqq p(i) \cdot G$ and sends $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}_i, p(i), T_i)$ to $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$, using a fresh, unique value for $\mathsf{id}^{\mathsf{com\text{-}zk}}_i$.

5. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ releases its proof by sending $(\texttt{decom-proof}, \mathsf{id}_i^{\mathsf{com\text{-}zk}})$ to $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$.

6. Each party $\mathcal{P}_i$ receives $(\texttt{accept}, \mathsf{id}_j^{\mathsf{com\text{-}zk}}, T_j)$ from $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$ for each $j \in [1,n] \setminus \{i\}$ if $\mathcal{P}_j$'s proof of knowledge is valid. $\mathcal{P}_i$ aborts if it receives $(\texttt{fail}, \mathsf{id}_j^{\mathsf{com\text{-}zk}})$ instead for any proof, or if there exists an index $x \in [1, n-t-1]$ such that $\mathbf{J}^x = [x, x+t]$ and $\mathbf{J}^{x+1} = [x+1, x+t+1]$ and

$$\sum_{j \in \mathbf{J}^x} \lambda_j^{\mathbf{J}^x}(0) \cdot T_j \neq \sum_{j \in \mathbf{J}^{x+1}} \lambda_j^{\mathbf{J}^{x+1}}(0) \cdot T_j$$

where $\lambda_j^{\mathbf{J}^x}(y)$ and $\lambda_j^{\mathbf{J}^{x+1}}(y)$ are party $\mathcal{P}_j$'s Lagrange coefficients for interpolating p at location y with the sets of parties indexed by $\mathbf{J}^x$ and $\mathbf{J}^{x+1}$ respectively.

7. The parties compute the shared public key using any subset $\mathbf{J} \subseteq [1,n]$ such that $|\mathbf{J}| = t$

$$\mathsf{pk} := \sum_{j \in \mathbf{J}} \lambda_j^{\mathbf{J}}(0) \cdot T_j$$

Auxiliary Setup:

8. Every party sends $(\texttt{init})$ to the $\mathcal{F}_{\mathsf{Inv}}^{n,t}$.

9. Every pair of parties $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $i < j$ sends $(\texttt{init})$ to $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$.

Once the a group of n parties has completed the $\pi_{\mathsf{ECDSA\text{-}Setup}}^{n,t}$ protocol, any t-sized subgroup of those parties indexed by $\mathbf{P}$ can run the following signing protocol to produce a signature.

Protocol 4. t-of-n ECDSA Signing $\left(\pi_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}}\right)$:

This protocol is parameterized by the party count n, the threshold size t, the group of parties $\mathbf{P} \subseteq [1,n]$ among which it runs, the elliptic curve $(\mathbb{G}, G, q)$, and the statistical security parameter s. It invokes the $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, $\mathcal{F}_{\mathsf{Inv}}^{n,t}$, and $\mathcal{F}_{\mathsf{Com}}^{n}$ functionalities. It takes as input from each party $\mathcal{P}_i$ for $i \in \mathbf{P}$ the public key pk, the message m, the signature index $\mathsf{id}^{\mathsf{sig}}$ (which is used to generate other unique indices as required), and the point $p(i)$ on the polynomial that encodes the secret key, and yields to each party a copy of the signature σ.

Multiplication and Inversion:

1. Each party $\mathcal{P}_i$ invokes $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ with a fresh, agreed-upon inversion index, and receives as output (u_i, v_i, R).

2. Each party $\mathcal{P}_i$ computes $\lambda_i^{\mathbf{P}}(0)$, its Lagrange coefficient given that it is reconstructing sk with the parties in $\mathbf{P}$. $\mathcal{P}_i$ then computes sk_i, its additive

share of the secret key for this group of parties

$$\mathsf{sk}_i := \lambda_i^{\mathbf{P}}(0) \cdot p(i)$$

3. Each pair of parties, $\mathcal{P}_i$ and $\mathcal{P}_j$ invoke $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ with $\ell = 2$. The party with the lower index plays the role of Alice and the other the role of Bob, and they use a fresh, agreed-upon multiplication index. The parties run the multiplication preprocessing and input phases, with $\mathcal{P}_i$ supplying as input $\{\mathsf{sk}_i, v_i\}$ and $\mathcal{P}_j$ supplying $\{v_j, \mathsf{sk}_j\}$. As outputs they receive $\{w_i^{j,1}, w_i^{j,2}\}$ and $\{w_j^{i,1}, w_j^{i,2}\}$, respectively. We again elide the specific messages involved in this process, but note that

$$w_i^{j,1} + w_j^{i,1} = \mathsf{sk}_i \cdot v_j \qquad \text{and} \qquad w_i^{j,2} + w_j^{i,2} = \mathsf{sk}_j \cdot v_i$$

4. Each party $\mathcal{P}_i$ sets

$$w_i := \mathsf{sk}_i \cdot v_i + \sum_{j \in \mathbf{P} \setminus \{i\}} \left(w_i^{j,1} + w_i^{j,2} \right)$$

Consistency Check:

5. Each party $\mathcal{P}_i$ calculates

$$\Gamma_i^2 := v_i \cdot \mathsf{pk} - w_i \cdot G \qquad \text{and} \qquad \Gamma_i^3 := w_i \cdot R$$

and commits to both values by sending $(\mathtt{commit}, \mathsf{id}_i^{\mathsf{com}}, \left(\Gamma_i^2, \Gamma_i^3\right), \mathbf{P})$ to $\mathcal{F}_{\mathsf{Com}}^n$, using a fresh value for $\mathsf{id}_i^{\mathsf{com}}$.

6. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ sends $(\mathtt{decommit}, \mathsf{id}_i^{\mathsf{com}})$ to $\mathcal{F}_{\mathsf{Com}}^n$ and collects $\{(\Gamma_j^2, \Gamma_j^3)\}_{j \in \mathbf{P} \setminus \{i\}}$ as the other parties do the same.

7. The parties abort if

$$\sum_{j \in \mathbf{P}} \Gamma_j^2 \neq 0 \qquad \text{or} \qquad \sum_{j \in \mathbf{P}} \Gamma_j^3 \neq \mathsf{pk}$$

Signing:

8. Each party $\mathcal{P}_i$ calculates

$$\mathsf{sig}_i := H(m) \cdot v_i + r_x \cdot w_i$$

and broadcasts sig_i.

9. Each party computes

$$\mathsf{sig} := \sum_{i \in \mathbf{P}} \mathsf{sig}_i \qquad \text{and} \qquad \sigma := (\mathsf{sig}, r_x)$$

where $(r_x, r_y) = R$, and outputs σ if $\mathsf{Verify}(\mathsf{pk}, \sigma) = 1$.

Our proof of security for the foregoing protocols relies on the Computational Diffie-Hellman Assumption in the group over which the signature is computed. Because this assumption is formally defined over an infinite sequence of groups, our security theorem must also consider such a sequence.

Theorem 5.1. *Let $(\mathbb{G}, \mathbf{G}, \mathbf{q})$ be an infinite sequence of elliptic curves in which the Computational Diffie-Hellman Problem is hard. The protocols $\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$ and $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ jointly UC-realize the functionality $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ for this sequence in the $(\mathcal{F}^{n,t}_{\mathsf{Inv}}, \mathcal{F}^{\ell}_{\mathsf{2PMul}}, \mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}, \mathcal{F}^{n}_{\mathsf{Com}})$-hybrid model, in the presence of a malicious adversary statically corrupting up to $t-1$ parties.*

Proof. A proof of this theorem is given in Appendix D. □

Setup Round Count. The Public Key Generation portion of $\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$ requires three broadcast rounds in total, but the initialization procedures in the Auxiliary Setup phase require five, when $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ is realized by Keller et al.'s OT-extension [KOS15] and the VSOT protocol [DKLs18], as we intend. Since Auxiliary Setup is independent of Key Generation, these phases can be run concurrently, and the round count can be as low as five, concretely. Our implementation, however, runs them in sequence, yielding eight concrete rounds.

Signing Optimizations and Round Count. For the sake of readability and ease of proof we have separated our threshold ECDSA signing system as cleanly as possible into independent functionalities and protocols. If we blur the boundaries between them slightly, we can achieve a smaller round count and reduced concrete costs with no loss in security. Consider the composite system represented by $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ when we realize $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ and $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ via $\pi^{n,t}_{\mathsf{Inv}}$ and $\pi^{\ell}_{\mathsf{2PMul}}$ respectively; in this composite system, three major optimizations are available.

1. Each party's inputs to the instances of $\pi^{\ell}_{\mathsf{2PMul}}$ contained within $\pi^{n,t}_{\mathsf{Inv}}$ are information-theoretically hidden from that party's counterparties prior to the multiplications themselves. This mitigates the potential attack on a two-round variant of $\pi^{\ell}_{\mathsf{2PMul}}$ mentioned at the end of Section 3, and consequently we can optimize $\pi^{\ell}_{\mathsf{2PMul}}$ by sending the adjustment messages simultaneously with the randomized multiplication, reducing each instance of $\pi^{\ell}_{\mathsf{2PMul}}$ to two rounds, and $\pi^{n,t}_{\mathsf{Inv}}$ to $\lceil \log(t) \rceil + 5$ rounds.

2. Each pair of parties initializes two separate instances of $\pi^{\ell}_{\mathsf{2PMul}}$ with $\ell = 2$ - one due to $\pi^{n,t}_{\mathsf{Inv}}$ and the other due to $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. These can be merged into a single instance with $\ell = 4$. Although the specifications of $\pi^{\ell}_{\mathsf{2PMul}}$ and $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ do not describe it explicitly, it is possible for Alice and Bob

run the Randomized Multiplication phase of $\pi^{\ell}_{\mathsf{2PMul}}$ for an entire batch, and then run the Input and Adjustment phase for elements in the batch individually, suppling inputs (and receive outputs) one element at a time as the multiplications are needed. This optimization removes the security-parameter-dependent overhead associated with evaluating a second batched multiplication during each signature. Such overhead is incurred both by $\pi^{\ell}_{\mathsf{2PMul}}$ itself, and by the protocol that instantiates $\mathcal{F}^{\eta}_{\mathsf{COTe}}$.

3. Both $\pi^{n,t}_{\mathsf{Inv}}$ and $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ have Consistency Check phases: in the course of $\pi^{n,t}_{\mathsf{Inv}}$ the parties each transmit a check value Γ^1_i, and in $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ they each transmit check values Γ^2_i and Γ^3_i. As described, the Consistency Check phase of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ cannot begin until $\pi^{n,t}_{\mathsf{Inv}}$ is complete, but minor changes allow us to transmit these values simultaneously. Specifically, if each party $\mathcal{P}_i$ replaces v_i with $\tilde{v}_i$ in their input to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}/\pi^{\ell}_{\mathsf{2PMul}}$ in Step 3 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, then this step can be evaluated immediately upon the completion of Step 5 of $\pi^{n,t}_{\mathsf{Inv}}$, before R is available. Step 4 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ will now yield for each party $\mathcal{P}_i$ a value $\tilde{w}_i$ such that

$$\sum_{i\in\mathbf{P}} \tilde{w}_i = \sum_{i\in\mathbf{P}} v_i \cdot \sum_{i\in\mathbf{P}} \mathsf{sk}_i \cdot \prod_{i\in\mathbf{P}} \phi_i = \frac{\phi\cdot\mathsf{sk}}{k}$$

Since $\pi^{\ell}_{\mathsf{2PMul}}$ requires only one round once the input becomes available, $\tilde{w}_i$ becomes available no later than R, and when R and $\tilde{w}_i$ are both known, $\mathcal{P}_i$ can compute three check values

$$\Gamma^1_i \coloneqq \tilde{v}_i \cdot R \qquad \text{and} \qquad \Gamma^2_i \coloneqq \tilde{v}_i \cdot \mathsf{pk} - \tilde{w}_i \cdot G \qquad \text{and} \qquad \Gamma^3_i \coloneqq \tilde{w}_i \cdot R$$

and commit to all three at once. Once the commitments are received, the parties decommit them along with their shares of ϕ and abort if

$$\sum_{i\in\mathbf{P}} \Gamma^1_i \neq \phi \cdot R \qquad \text{or} \qquad \Gamma^2_i \neq 0 \qquad \text{or} \qquad \Gamma^3_i \neq \phi\cdot\mathsf{pk}$$

Finally, each party $\mathcal{P}_i$ computes $v_i \coloneqq \tilde{v}_i/\phi$ and $w_i \coloneqq \tilde{w}_i/\phi$, and continues with the signing procedure as described in Steps 8 and 9 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. This optimization allows the GMW-style multiplier and Consistency Check components of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ to be run in parallel with $\pi^{n,t}_{\mathsf{Inv}}$, and as a result they contribute no extra rounds to the composite protocol.

Given these optimizations, $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ requires only a single additional round over those required by $\pi^{n,t}_{\mathsf{Inv}}$, in which shares of the final signature are exchanged. Since $\pi^{n,t}_{\mathsf{Inv}}$ requires only $\lceil\log(t)\rceil + 5$ rounds in the context of our protocol, the total round count for ECDSA signing comes to $\lceil\log(t)\rceil + 6$.

6 Cost Analysis

In Tables 1 and 2 we provide accountings of the communication and computation costs respectively for the four protocols that we have presented. Round counts

Offline (Setup) Costs			
Protocol	Rounds	Party	Bits Transmitted Per Party
$\pi^{\ell}_{\mathsf{2PMul}}$	5	Alice	$2\kappa^2 + \kappa$
		Bob	$3\kappa^2 + 3\kappa + 2$
$\pi^{n,t}_{\mathsf{Inv}}$	5	$\mathcal{P}_1$	$(n-1)\cdot(2\kappa^2 + \kappa)$
		$\mathcal{P}_n$	$(n-1)\cdot(3\kappa^2 + 3\kappa + 2)$
$\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$	5	$\mathcal{P}_1$	$(n-1)\cdot(2\kappa^2 + 6\kappa + 2)$
		$\mathcal{P}_n$	$(n-1)\cdot(3\kappa^2 + 8\kappa + 4)$
Online Costs			
Protocol	Rounds	Party	Bits Transmitted Per Party
$\pi^{\ell}_{\mathsf{2PMul}}$	3	Alice	$(2\ell+1)\cdot\kappa^2 + (4\ell+2)\cdot\kappa\cdot s + 2\ell\cdot\kappa$
		Bob	$\ell\cdot\kappa^2 + (2\ell+1)\cdot\kappa\cdot s + (\ell+130)\kappa$
$\pi^{n,t}_{\mathsf{Inv}}$	$\lceil\log(t)\rceil + 6$	$\mathcal{P}_{\mathbf{P}_1}$	$(t-1)\cdot(5\kappa^2 + 10\kappa\cdot s + 10\kappa + 2)$
		$\mathcal{P}_{\mathbf{P}_t}$	$(t-1)\cdot(2\kappa^2 + 5\kappa\cdot s + 138\kappa + 2)$
$\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$	$\lceil\log(t)\rceil + 6$	$\mathcal{P}_{\mathbf{P}_1}$	$(t-1)\cdot(9\kappa^2 + 18\kappa\cdot s + 18\kappa + 4)$
		$\mathcal{P}_{\mathbf{P}_t}$	$(t-1)\cdot(4\kappa^2 + 9\kappa\cdot s + 144\kappa + 4)$

Table 1: **Communication Cost Equations for Subprotocols**. In this table we do not consider the round-reducing optimizations for $\pi^{\ell}_{\mathsf{2PMul}}$ and $\pi^{n,t}_{\mathsf{Inv}}$, but we do consider all of the available optimizations for $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. Note that we count the number of bits transmitted by a single party to all other parties.

for our protocols are discussed at length in the relevant sections. Our equations assume that elements from $\mathbb{Z}_q$ are represented in κ bits, and that curve points are transmitted with point compression and thus are represented in $\kappa + 1$ bits. We assume that commitments require a single hash to generate and the transmission of a single element from $\mathbb{Z}_q$, that decommitments consist simply of the committed values and require a single hash to verify, and that zero-knowledge proofs of knowledge of discrete logarithm are implemented with Schnorr protocols and the Fiat-Shamir heuristic, and thus comprise a curve point and an element from $\mathbb{Z}_q$, along with the point for which knowledge of discrete logarithm is to be proven, and require one elliptic curve multiplication and one hash from the prover, and two curve multiplications plus one hash from the verifier. Finally, we assume that $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ functionality is realized via the protocol of Keller et al. [KOS15] with the VSOT protocol [DKLs18] supplying base-OTs. Keller et al.'s protocol requires an additional security parameter, which we set as $128 + s$, following their analysis. Concretely, for $\kappa = 256$ and $s = 80$, the average total number of bits transmitted per party is $(t-1)\cdot 88.28$ KiB, and for setup $(n-1)\cdot 20.22$ KiB. As an example, average costs for $n = 16$ and $t = 8$ are 303 KiB transmitted per party for setup, and 618 KiB for signing.

We observe that among the contemporaries of this work [LN18, GG18], ours is

Offline (Setup) Costs			
Protocol	Party	EC Multiplications	Hash Invocations
$\pi^{\ell}_{\mathsf{2PMul}}$	Alice	5	6
	Bob	4	7
$\pi^{n,t}_{\mathsf{Inv}}$	$\mathcal{P}_1$	$5n-5$	$6n-6$
	$\mathcal{P}_n$	$4n-4$	$7n-7$
$\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$	$\mathcal{P}_1$	$t \cdot n - t^2 + 7n - t - 5$	$8n-6$
	$\mathcal{P}_n$	$t \cdot n - t^2 + 6n - t - 4$	$9n-7$
Online Costs			
Protocol	Party	EC Multiplications	Hash Invocations
$\pi^{\ell}_{\mathsf{2PMul}}$	Alice	0	$6\ell \cdot \kappa + 12\ell \cdot s + 2\ell + 2s + 256$
	Bob	0	$5\ell \cdot \kappa + 10\ell \cdot s + 2\ell + 3s + 384$
$\pi^{n,t}_{\mathsf{Inv}}$	$\mathcal{P}_{\mathbf{P}_1}$	3	$3t + (t-1) \cdot (12\kappa + 26s + 260)$
	$\mathcal{P}_{\mathbf{P}_t}$	3	$3t + (t-1) \cdot (10\kappa + 23s + 388)$
$\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$	$\mathcal{P}_{\mathbf{P}_1}$	6	$4t + (t-1) \cdot (24\kappa + 50s + 264)$
	$\mathcal{P}_{\mathbf{P}_t}$	6	$4t + (t-1) \cdot (20\kappa + 43s + 392)$

Table 2: **Computation Cost Equations for Subprotocols**. In this table we consider all of the available optimizations for $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$.

the only signing protocol with a superconstant round count,[2] but also the only one with a constant number of public-key operations per signature. The benchmarks we report in Section 7 reveal our approach to have yielded competitive concrete performance, even in the context of large numbers of parties and high latency. Furthermore, we observe that the public key operations required by other signing protocols are incurred in part by non-interactive zero-knowledge proofs. If UC-security is to be achieved in an implementation of any protocol, the Fiat-Shamir transform must be abandoned in favor of an alternative with straight-line extraction, such as Fischlin's transform [Fis05]. Such straight-line extractable transforms incur a much larger practical cost than Fiat-Shamir. Our signing protocol makes no black-box use of zero-knowledge functionalities (instead relying on an algebraic check), and so we avoid this UC-security penalty completely.

7 Implementation

We created a proof-of-concept implementation of our protocols in the Rust language, which is derived from the open source 2-of-n implementation of Doerner et al. [DKLs18]. Our implementation uses the secp256k1 curve, as standardized by NIST [Bro10]. Thus, for all benchmarks, $\kappa = 256$, and we set

[2] As mentioned in footnote 1, a constant-round version of our protocol is in development.

the statistical security parameter $s = 80$. We instantiated the $\mathcal{F}_{\mathsf{COTe}}^{\eta}$ functionality using the protocol of Keller et al. [KOS15] and set the OT-extensions security parameter to $128 + s$, following their analysis. We chose, as Doerner et al. did, to instantiate $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$ via the Fiat-Shamir Heuristic (though we note that this transform is *not* UC-secure), and to instantiate the PRG, the random oracle H, and the commitment functionality $\mathcal{F}_{\mathsf{Com}}^{n}$ via SHA-256. Thus, our protocol uses the same concrete hash function as specified in the ECDSA standard. We note that while the folkloric hash-based instantiation of $\mathcal{F}_{\mathsf{Com}}^{n}$ (i.e. $H(m\|r)$ where m is the message, and $r \leftarrow \{0,1\}^{\kappa}$) requires a uniform nonce in order to hide the message regardless of its distribution, in our protocol all committed messages have sufficient entropy that the nonce can be omitted.

Unlike Doerner et al., we do not parallelize vectors of hashing operations. Instead, each party parallelizes its interactions with its counterparties (and the computations that they require), using a number of threads equal to the number of parties, or a specified maximum, whichever is smaller. While we have assumed throughout this paper that the setup protocol can parallelize key-generation and OT-extension initialization, our implementation runs these two phases sequentially, and thus the practical round count is eight, whereas the theoretical round count is five.

We benchmarked our implementation using a set of Google Cloud Platform `n1-highcpu-8` nodes, each running Ubuntu 18.04 with kernel 4.15.0. Each node of this type has a CPU from the Intel Skylake family, with four physical cores clocked at 2.0 GHz, capable of executing eight threads simultaneously in total. These machines are slightly slower than those used by Doerner et al. [DKLs18], and thus the timings we report for their protocol are slightly slower than they report themselves. Each party participating in a benchmark was allocated one node, and the parties communicated via Google's internal network. We compiled our code using the nightly version of Rust 1.28, with the default level of optimization. Parallelism was provided by the Rayon crate and, as each node can execute eight threads simultaneously, we limited the number of threads used in signing to ten (having arrived at this number empirically). Our hash function implementations were written in C using compiler intrinsics, and were compiled with GCC 8.2.0. Our benchmarking programs were designed to establish insecure connections among the parties one time only, and then run a batch of setup or signing operations, measuring the wall clock time for the entire batch. Thus, they record overhead due to latency and bandwidth constraints, but they do not record overhead due to private or authenticated channels.

7.1 LAN Benchmarks

For benchmarks in the LAN setting, we created a set of 256 nodes in Google's South Carolina datacenter. Among these nodes, we measured the bandwidth to be generally between 5 and 10 Gbits/sec, and the round-trip latency to be approximately 0.3 ms. Using these nodes, we collected data for both our setup and signing protocols using combinations of parameters as specified in Table 3. For signing benchmarks, all costs are independent of n, the number of parties in

n/t Range	n/t Step	Samples (Signing)	Samples (Setup)
$[2, 8]$	1	16000	2000
$(8, 16]$	2	8000	1000
$(16, 32]$	4	4000	500
$(32, 64]$	8	2000	250
$(64, 128]$	16	1000	125
$(128, 256]$	32	500	62

Table 3: **LAN Benchmark Parameters**. For signing we varied t according to these parameters, and for setup we varied n, fixing $t = \lceil n/2 \rceil$.

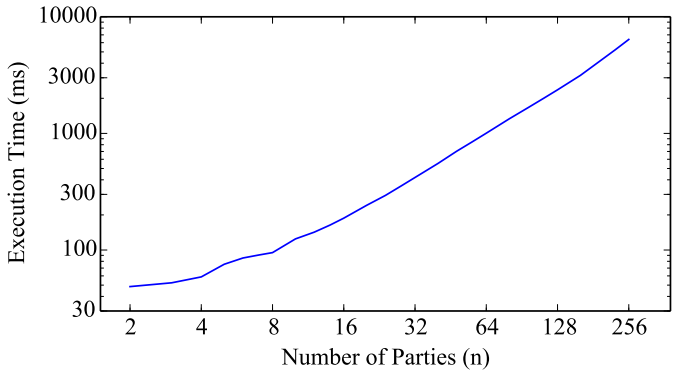

Figure 1: **Wall Clock Times for n-Party Setup over LAN**. Note that all parties reside on individual machines in the same datacenter, and latency is on the order of a few tenths of a millisecond.

the larger group from whom the signing parties are selected. Consequently, we varied only t, the number of parties actually participating in signing. For setup, only computation costs depend upon t, and not bandwidth; consequently we varied n and set $t = \lceil n/2 \rceil$, which we determined to be the most expensive value relative to a particular choice of n. Our aim in choosing sample counts was to ensure each benchmark took five to ten minutes in total, in order to smooth out artifacts due to transient network conditions. Our results for setup are reported in Figure 1, and our results for signing are reported in Figure 2.

7.2 Comparison

We note that our method only slightly underperforms that of Doerner et al. [DKLs18] for 2-of-n signing, in spite of the fact that our protocol implements a somewhat stronger functionality. Specifically, we require 8.6 ms, whereas an evaluation of their protocol (with no parallelism) in our benchmarking environ-

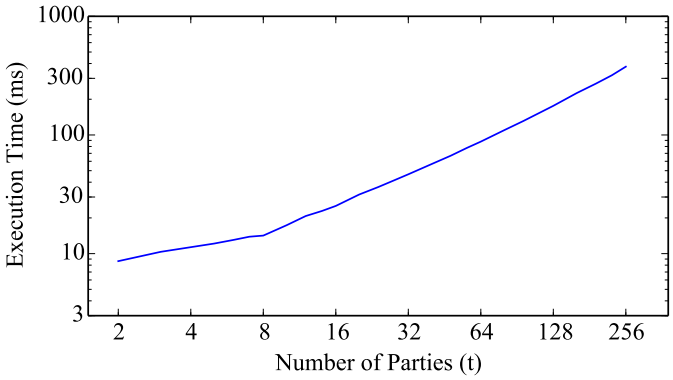

Figure 2: **Wall Clock Times for t-Party Signing over LAN**. Note that all parties reside on individual machines in the same datacenter, and latency is on the order of a few tenths of a millisecond.

	Signing		Setup	
Protocol	$t = 2$	$t = 20$	$n = 2$	$n = 20$
This Work	8.6	31.5	48.5	242
[GG18]	77	509	–	–
[LNR18]	304	5194	~11000	~28000
[BGG17]	~650	~1500	–	–
[GGN16]	205	1136	–	–
[Lin17]	36.8	–	2435	–
[DKLs18]	3.8	–	43.4	177

Table 4: **Wall-clock Time Comparison to Other Works**. Note that benchmarking environments are non-identical, but all benchmarks are networkless or over LAN.

ment requires 5.8 ms. In the arbitrary-threshold context, a number of prior and concurrent works exist. We did not benchmark their protocols in our environment, and so no truly fair comparison is possible. Nevertheless, all of them report benchmarks among 2 to 20 LAN-connected parties on hardware broadly similar to our own, and we believe it possible to draw some loose conclusions by comparing their results. We reproduce setup and signing times from a selection of publications in Table 4.

The protocol of Gennaro and Goldfeder [GG18] appears to be the fastest prior or concurrent work for signing, although they do not report benchmarks for their key-generation protocol. Their benchmarks were performed using 3.4 GHz processors from the Intel Skylake family, but they used only a single thread and did not count network costs. In another concurrent work, Lindell et

al. [LNR18] present a different protocol and perform benchmarks using reasonably recent 2.4 GHz processors from the Intel Haswell family. Their benchmarks do count network costs, but like Gennaro and Goldfeder, they use only a single thread. Among prior works, the most efficient techniques are those of Gennaro et al. [GGN16] and Boneh et al. [BGG17] (who provide an improved implementation Gennaro et al.'s protocol in addition to developing new techniques). Boneh et al. provide benchmarks for both protocols, with no network costs recorded. In all parameter regimes reported, all prior and concurrent works are at least one order of magnitude slower than our own in terms of both key-generation and signing, and in some cases we improve upon them by two or more orders of magnitude. We stress again that as these benchmarks were not run in identical environments, they do not constitute a fair comparison. Nevertheless, we do not believe that environmental differences account for the performance discrepancy.

7.3 WAN Benchmarks

As we have previously noted, our protocol is at a disadvantage relative to other approaches in terms of round count. In order to demonstrate the practical implications of this fact, we ran an additional benchmark in the WAN setting. We chose 16 Google datacenters (otherwise known as *zones*) that offer instances with current-generation CPUs; these are located on a map in Figure 3. Five were located inside the United States, in South Carolina, Virginia, Oregon, California, and Iowa. Among these, the longest leg was between Oregon and South Carolina, with a round-trip latency of 66.5 ms and bandwidth of 353 Mbits/sec. The remaining 11 were located in Montreál, London, Frankfurt, Belgium, the Netherlands, Finland, Sydney, Taiwan, Tokyo, Mumbai, and Singapore. Among the complete set, the longest leg was between Belgium and Mumbai, with a round-trip latency of 348 ms and a bandwidth of 53.4 MBits/sec. We tested two configurations: one with only the five US datacenters participating, and another with all 16. For each configuration, we performed benchmarks with one party in each participating datacenter, and with sixteen parties in each participating datacenter. In all cases, we collected 125 samples. Results are reported in Table 5, along with comparative data from our LAN benchmarks. As we mentioned, the protocol of Gennaro and Goldfeder [GG18] appears to be the fastest prior or concurrent work for signing; we observe that with 20 parties and no latency, their protocol is slightly slower than ours is with 80 parties spread across the United States.

It is worth noting that Wang et al. [WRK17] recently made the claim that they performed the largest-scale demonstration of multiparty computation to date. Their benchmark involves 128 parties split among eight datacenters around the world, who jointly compute an AES circuit using the actively-secure multiparty garbling protocol that they developed. Our WAN benchmark involves 256 parties split among 16 datacenters, and thus we seem also to have evaluated one of the largest secure multiparty protocols to date, at least so far as party count and geographic distribution are concerned. We also note that the in the clear setting, AES is generally considered to have a much lower circuit complexity than ECDSA;

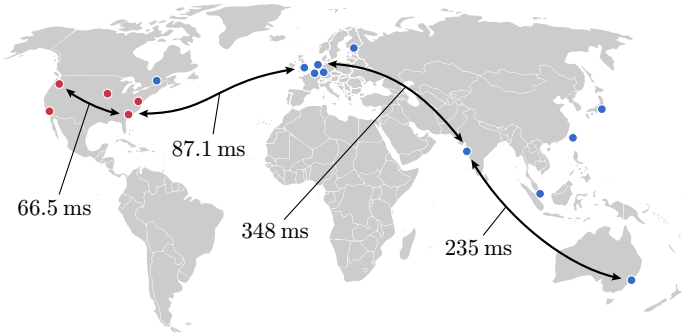

Figure 3: **Map of Datacenter Locations used for WAN Benchmarks**, with latency figures along a few of the longer routes. The subgroup of five zones inside the US are highlighted in red.

Parties/Zones	Signing Rounds	Signing Time	Setup Time
5/1	9	12.2	67.9
5/5	9	288	330
16/1	10	25.2	181
16/16	10	3018	1655
80/1	12	109	539
80/5	12	388	1521
256/1	13	376	2300
256/16	13	4408	7283

Table 5: **Wall-clock Times in Milliseconds over WAN**. The benchmark configurations used are described in Section 7.3. For signing we varied t according to these parameters, and for setup we varied n, fixing $t = \lfloor (n+1)/2 \rfloor$. Benchmarks involving only a single zone are LAN benchmarks, for comparison.

this is reflected in the significantly lower computation time for a single AES operation as compared to signing a single message using ECDSA. Interestingly, in the context of evaluating these primitives securely among multiple parties, our protocol for realizing $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$ performs considerably better than Wang et al.'s realization of $\mathcal{F}_{\mathsf{AES}}^{n}$. In the LAN setting with 128 parties (each much more powerful than the ones we employ), they report a 17-second wall clock time, including preprocessing, and in the global WAN setting with 128 parties, their protocol requires 2.5 minutes. When the setup and signing costs are combined for our protocol, it requires 6.8 seconds and 11.7 seconds with 256 parties in the LAN and global WAN settings, respectively. This discrepancy in performance is counterintuitive, but unsurprising in light of the fact that the algebraically structured nature of ECDSA allows custom protocols such as our own to be devised. We believe that this serves to demonstrate that there are multiparty

Configuration	Benchmark	Setup Time	Signing Time
Macbook/RPi	2-of-2	1308	49.8
2×RPi	2-of-2	1320	55.6
2×RPi	2-of-n	–	66.6
3×RPi	3-of-3	1390	92.8

Table 6: **Wall-clock Times in Milliseconds for Raspberry Pi**. The benchmark configurations used are described in Section 7.4.

functionalities for which specially tailored protocols are warranted in practice, as opposed to the blind use of generic MPC for all tasks.

7.4 Low-power Benchmarks

Finally, we performed a set of benchmarks on a group of three Raspberry Pi model 3B+ single-board computers in order to demonstrate the feasibility of evaluating our protocol (and the protocols of Doerner et al. [DKLs18]) on small, low-powered devices. Each board has a single, quad-core ARM-based processor clocked at 1.4 GHz. The boards were loaded with Raspbian Linux (kernel 4.14) and connected to one another via ethernet. As an optimization for the embedded setting, we abandoned SHA-256 (except where required by ECDSA) in favor of the BLAKE2 hash function [ANWOW13], using assembly implementations provided by the BLAKE2 authors. To simulate the setting wherein an embedded device signs with a more powerful one, we used a 2018 15" Macbook Pro running Mac OS 10.14 (i.e. one author's laptop). This machine was engaged in other tasks at the time of benchmarking, and no attempt was made to prevent this. We benchmarked 2-of-2 signing and setup between the Macbook and a single Raspberry Pi, and t-of-n setup and signing among the group of Pis, with n set as 3 and t as both 2 and 3. When $n = 2$, we used the slightly more efficient protocols of Doerner et al. [DKLs18] without modification, and when $t = 3$ we used the protocols presented in this paper. For setup, we collected 50 samples, and for signing, we collected 250. Results are presented in Table 6. We observe that in spite of the limitations of the hardware on which these benchmarks were run, the signing time remains much less than a second, and setup requires only a few seconds. Thus we expect our protocol to be computationally efficient enough to run even on embedded devices such as hardware tokens or smartwatches, and certainly on more powerful mobile devices such as phones.

8 Code Availability

Our implementation is available under the three-clause BSD license from `https://gitlab.com/neucrypt/mpecdsa`.

9 Acknowledgments

We thank Dennis Giese for providing the hardware used in our low-power device benchmark, and Ran Cohen for many helpful discussions and useful advice. The authors of this work are supported by NSF grant TWC-1664445 and a Google Faculty fellowship award.

$\ddot{\pi}$

References

[Ame05] American National Standards Institute. X9.62: Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.

[ANWOW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. `https://blake2.net/blake2.pdf`, 2013.

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, 1989.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, 1996.

[BGG17] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In *LATINCRYPT*, 2017.

[Bit17] Bitcoin Wiki. Transaction. `https://en.bitcoin.it/wiki/Transaction`, 2017. Accessed Oct 22, 2017.

[Bro05] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptography*, 2005.

[Bro10] Daniel R. L. Brown. Sec 2: Recommended elliptic curve domain parameters, 2010.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical uc security with a global random oracle. In *CCS*, 2014.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *LATINCRYPT*, 2015.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO*, 1987.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 1976.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *IEEE S&P*, 2018.

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

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *CCS*, 2018.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. *Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security*. 2016.

[Gil99] Niv Gilboa. Two party rsa key generation. In *CRYPTO*, 1999.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *STOC*, 1989.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptol.*, 1996.

[Jou04] Antoine Joux. A one round protocol for tripartite diffie-hellman. *J. Cryptol.*, 2004.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*, chapter Digital Signature Schemes, pages 443–486. Chapman & Hall/CRC, 2015.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *CRYPTO*, 2015.

[Kra93] D.W. Kravitz. Digital signature algorithm, jul 1993. US Patent 5,231,668.

[Lin17] Yehuda Lindell. Fast secure two-party ecdsa signing. In *CRYPTO*, 2017.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, 2018.

[LNR18] Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Report 2018/987, 2018. `https://eprint.iacr.org/2018/987`.

[MR01] Philip MacKenzie and Michael K. Reiter. Two-party generation of dsa signatures. In *CRYPTO*, 2001.

[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.

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

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

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 1979.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[Woo17] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2017.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, 2017.

A Introduction to our Proofs

In the appendices to follow, we prove the protocols from the foregoing paper secure against a malicious adversary in the Universal Composability (UC) framework. For a full introduction to UC, we refer the reader to Canetti [Can01]. Let $\mathcal{Z}$ denote the environment, and let $\mathcal{A}^{\mathbf{P}^*}$ denote an adversary who corrupts a subset of parties indexed by $\mathbf{P}^*$. Suppose we have an n-party protocol π with a corresponding ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}^{\mathbf{P}^*}$ that interacts the corrupt parties, and suppose we denote by $\mathbf{C}$ the set of all subsets of parties against which we wish to prove π secure (i.e. the set of all possible values of $\mathbf{P}^*$).

We define the *real-world* experiment $\mathsf{REAL}_{\pi,\mathcal{A}^{\mathbf{P}^*},\mathcal{Z}}(z)$ to proceed in the following way: the environment receives some nonuniform advice z and determines the inputs of all n parties, who then engage in the protocol π, those parties indexed by $\mathbf{P}^*$ acting according to the corrupt instructions of $\mathcal{A}^{\mathbf{P}^*}$, and the others following the protocol honestly; after the interaction of the parties is complete, $\mathcal{Z}$ receives the outputs of all parties, plus an output from $\mathcal{A}^{\mathbf{P}^*}$ that is characterized by the messages received by the parties indexed by $\mathbf{P}^*$, and using this information $\mathcal{Z}$ outputs a bit, which is the result of the experiment.

In the *ideal-world* experiment $\mathsf{IDEAL}_{\mathcal{F},(\mathcal{S}^{\mathbf{P}^*},\mathcal{A}^{\mathbf{P}^*}),\mathcal{Z}}(z)$, on the other hand, the interaction of the parties is defined by the ideal functionality $\mathcal{F}$. The honest parties are replaced by ideal counterparts, and parties corrupted by $\mathcal{A}^{\mathbf{P}^*}$ interact with the simulator $\mathcal{S}^{\mathbf{P}^*}$, which interacts with $\mathcal{F}$ on their behalf. Thus the ideal adversary is defined by the tuple $(\mathcal{S}^{\mathbf{P}^*},\mathcal{A}^{\mathbf{P}^*})$. A proof that for any $\mathcal{A}^{\mathbf{P}^*}$,

$$\left\{\mathsf{REAL}_{\pi,\mathcal{A}^{\mathbf{P}^*},\mathcal{Z}}(z)\right\}_{z\in\{0,1\}^*,\mathbf{P}^*\in\mathbf{C}} \equiv \left\{\mathsf{IDEAL}_{\mathcal{F},(\mathcal{S}^{\mathbf{P}^*},\mathcal{A}^{\mathbf{P}^*}),\mathcal{Z}}(z)\right\}_{z\in\{0,1\}^*,\mathbf{P}^*\in\mathbf{C}}$$

is a proof that π *UC-realizes* $\mathcal{F}$ in the presence of a malicious adversary statically corrupting parties as given by $\mathbf{C}$, or, in other words, a proof of security for π.

Organization. In Appendix B, we prove that $\pi^{\ell}_{\mathsf{2PMul}}$ UC-realizes $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ in the presence of one statically corrupted party. In Appendix C, we prove that $\pi^{n,t}_{\mathsf{Inv}}$ UC-realizes $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ in the presence of $t-1$ statically corrupted parties. Finally, in Appendix D, we prove that $\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$ and $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ jointly UC-realize $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ in the presence of $t-1$ statically corrupted parties. All of these proofs share the same basic structure: we begin with a Security Theorem, give a simulator to serve as the ideal adversary in an ideal-world experiment, and then the proof proceeds as a sequence of hybrid experiments.

B Proof of Security for Two-party Multiplication

Theorem 3.1. *When $\ell \leq c \cdot \log(\kappa)$ for some constant c, the protocol $\pi^{\ell}_{\mathsf{2PMul}}$ UC-realizes the functionality $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ for a κ-bit field $\mathbb{Z}_q$ in the $\mathcal{F}^{n}_{\mathsf{COTe}}$-hybrid Global Random Oracle Model, in the presence of a malicious adversary statically corrupting either party.*

Proof. In Appendices B.1 and B.2 we prove respectively that there exist simulators $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ and $\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}$ such that over all $z \in \{0,1\}^*$,

$$\mathsf{REAL}_{\pi^{\ell}_{\mathsf{2PMul}},\mathcal{A}^{\mathsf{A}},\mathcal{Z}}(z) \overset{\mathsf{c}}{\equiv} \mathsf{IDEAL}_{\mathcal{F}^{\ell}_{\mathsf{2PMul}},(\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}},\mathcal{A}^{\mathsf{A}}),\mathcal{Z}}(z)$$
$$\mathsf{REAL}_{\pi^{\ell}_{\mathsf{2PMul}},\mathcal{A}^{\mathsf{B}},\mathcal{Z}}(z) = \mathsf{IDEAL}_{\mathcal{F}^{\ell}_{\mathsf{2PMul}},(\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}},\mathcal{A}^{\mathsf{B}}),\mathcal{Z}}(z)$$

The conjunction of these statements yields Theorem 3.1. □

B.1 Simulating Against Alice

We now present the simulation against Alice. We note that in Step 2 of $\pi^{\ell}_{\mathsf{2PMul}}$, we specify that Alice sends to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$

$$\boldsymbol{\alpha} \coloneqq \{\tilde{\mathbf{a}}_1 \| \hat{\mathbf{a}}_1\}_{j\in[1,\xi]} \,\|\, \ldots \,\|\, \{\tilde{\mathbf{a}}_\ell \| \hat{\mathbf{a}}_\ell\}_{j\in[1,\xi]}$$

for some vectors $\tilde{\mathbf{a}}, \hat{\mathbf{a}} \in \mathbb{Z}_q^{\ell}$, whereas in Step 2 of our simulator, we interpret the received value as

$$\left\{\tilde{\boldsymbol{\alpha}}_j \| \hat{\boldsymbol{\alpha}}_j\right\}_{j\in[1,\eta]} = \boldsymbol{\alpha}$$

using a different variable. We use this notational discrepancy to indicate that, though the protocol instructs Alice to send a vector $\boldsymbol{\alpha}$ containing repetitions of the same value, a malicious Alice may not actually do so, and thus values that should be identical may differ arbitrarily in the received vector.

Simulator 1. Two-party Multiplication against Alice $\left(\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}\right)$:

This simulator interposes between the ideal functionality $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and a malicious Alice running the $\pi^{\ell}_{\mathsf{2PMul}}$ protocol. It is parameterized by the statistical security parameter s and the symmetric security parameter κ, with $\xi = \kappa + 2s$ and $\eta = \xi \cdot \ell$. It also makes use of a gadget vector $\mathbf{g}$ of the same form as that used by $\pi^{\ell}_{\mathsf{2PMul}}$. It plays the role of the functionality $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ in its interaction with Alice, and it can observe Alice's queries to the random oracle H.

Init: Receive $(\texttt{init})$ from Alice on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ and send $(\texttt{init})$ to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

Multiplication:

1. Upon receiving $(\texttt{bob-ready}, \mathsf{id}^{\mathsf{mul}})$ from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, send $(\texttt{chosen}, \mathsf{id}^{\mathsf{ext}})$ to Alice on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, where $\mathsf{id}^{\mathsf{ext}}$ is a fresh index.

2. Upon receiving $(\texttt{transfer}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\alpha})$ from Alice on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, sample

   $$\tilde{\mathbf{z}}_{\mathsf{A}} \leftarrow \mathbb{Z}_q^{\eta} \qquad \text{and} \qquad \hat{\mathbf{z}}_{\mathsf{A}} \leftarrow \mathbb{Z}_q^{\eta}$$

   and let

   $$\left\{\tilde{\boldsymbol{\alpha}}_j \| \hat{\boldsymbol{\alpha}}_j\right\}_{j\in[1,\eta]} = \boldsymbol{\alpha}$$
   $$\boldsymbol{\omega}_{\mathsf{A}} = \left\{\tilde{\mathbf{z}}_{\mathsf{A},j} \middle\| \hat{\mathbf{z}}_{\mathsf{A},j}\right\}_{j\in[1,\eta]}$$

   and send $(\mathsf{pad}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}_{\mathsf{A}})$ to Alice on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$.

3. Engage in the coin tossing protocol (corresponding to Step 4 of $\pi^{\ell}_{\mathsf{2PMul}}$) with Alice to generate $\tilde{\boldsymbol{\chi}} \in \mathbb{Z}_q^{\ell}$ and $\hat{\boldsymbol{\chi}} \in \mathbb{Z}_q^{\ell}$. Using these values, compute $\mathbf{r}^*$, the value of $\mathbf{r}$ that Alice is expected to send if she is honest.

   $$\mathbf{r}^* \coloneqq \left\{ \sum_{i\in[0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j} \right\}_{j\in[1,\xi]}$$

4. Receive $\mathbf{u} \in \mathbb{Z}_q^\ell$ and $\mathbf{r} \in \mathbb{Z}_q^\xi$ from Alice, and compute the vector of values by which her inputs are offset from the expectation. Note that if Alice is honest, this vector will contain only zeros.

$$\boldsymbol{\Delta} \coloneqq \left\{ \tilde{\boldsymbol{\chi}}_{\lceil j/\ell \rceil} \cdot \tilde{\boldsymbol{\alpha}}_j + \hat{\boldsymbol{\chi}}_{\lceil j/\ell \rceil} \cdot \hat{\boldsymbol{\alpha}}_j - \mathbf{u}_{\lceil j/\ell \rceil} \right\}_{j \in [1,\eta]}$$

5. Check whether there exists a set of choice bits $\boldsymbol{\beta}^*$ that would allow Alice's cheats to go undetected in a real exectution of the protocol. That is, find $\boldsymbol{\beta}^* \in \{0,1\}^\eta$ such that

$$\bigwedge_{j \in [1,\xi]} \left\{ \mathbf{r}_j^* - \mathbf{r}_j = \sum_{i \in [0,\ell)} \boldsymbol{\beta}^*_{i \cdot \xi + j} \cdot \boldsymbol{\Delta}_{i \cdot \xi + j} \right\}$$

Since this condition comprises ξ individual predicates, each on ℓ bits, an exhaustive search can be performed in $O(\xi \cdot 2^\ell)$, where $\ell = O(\log \kappa)$. If no such $\boldsymbol{\beta}^*$ exists, then abort. If such a $\boldsymbol{\beta}^*$ does exist, then flip one coin for each nonzero value in $\boldsymbol{\Delta}$, and abort if any coin comes up heads.

6. Find $\tilde{\mathbf{a}} \in \mathbb{Z}_q^\ell$ such that

$$\bigwedge_{i \in [0,\ell)} \left\{ \begin{array}{l} \exists j \in [1,\xi] : \tilde{\mathbf{a}}_i = \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} \\ \quad \wedge \ \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j} = \mathbf{u}_{i+1} \end{array} \right\}$$

If no such $\tilde{\mathbf{a}}$ exists, or if more than one valid candidate for $\tilde{\mathbf{a}}$ exists, then abort. If exactly one candidate for $\tilde{\mathbf{a}}$ exists, then compute the additive offsets $\mathbf{d} \in \mathbb{Z}_q^\ell$ caused by any undetected cheating on the part of Alice

$$\mathbf{d} \coloneqq \left\{ \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \boldsymbol{\beta}^*_{i \cdot \xi + j} \cdot \left(\tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} - \tilde{\mathbf{a}}_{i+1} \right) \right\}_{i \in [0,\ell)}$$

and send $(\texttt{alice-ready}, \mathsf{id}^{\mathsf{mul}})$ to $\mathcal{F}_{\mathsf{2PMul}}^\ell$.

7. Wait for either a message from $\mathcal{F}_{\mathsf{2PMul}}^\ell$, or a message from Alice. If $\boldsymbol{\gamma}_{\mathsf{A}} \in \mathbb{Z}_q^\ell$ is received from Alice before anything is received from the functionality, then

   (a) Compute Alice's true input

$$\mathbf{a} \coloneqq \left\{ \tilde{\mathbf{a}}_i + \boldsymbol{\gamma}_{\mathsf{A},i} \right\}_{i \in [1,\ell]}$$

   and send $(\texttt{input}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a})$ to $\mathcal{F}_{\mathsf{2PMul}}^\ell$.

(b) On receiving $(\texttt{output}, \mathsf{id}^{\mathsf{mul}}, \mathbf{z}_{\mathsf{A}})$ from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, calculate the appropriate adjustment message. For $i \in [1, \ell]$, if $\mathbf{a}_i = 0$, then sample $\boldsymbol{\gamma}_{\mathsf{B},i} \leftarrow \mathbb{Z}_q$. Otherwise, set

$$\boldsymbol{\gamma}_{\mathsf{B},i} := \frac{\mathbf{z}_{\mathsf{A},i} + \mathbf{d}_i - \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},(i-1)\cdot\xi+j}}{\mathbf{a}_i}$$

and send $\boldsymbol{\gamma}_{\mathsf{B}}$ to Alice and terminate successfully.

On the other hand, if $(\texttt{bob-complete}, \mathsf{id}^{\mathsf{mul}})$ is received from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ before anything is received from Alice, then

(a) Sample and send $\boldsymbol{\gamma}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\ell}$ to Alice on behalf of Bob.

(b) On receiving $\boldsymbol{\gamma}_{\mathsf{A}} \in \mathbb{Z}_q^{\ell}$ from Alice, compute her true input and her output share

$$\mathbf{a} := \left\{ \tilde{\mathbf{a}}_i + \boldsymbol{\gamma}_{\mathsf{A},i} \right\}_{i \in [1,\ell]}$$

$$\mathbf{z}_{\mathsf{A}} := \left\{ \mathbf{a}_i \cdot \boldsymbol{\gamma}_{\mathsf{B},i} - \mathbf{d}_i + \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},(i-1)\cdot\xi+j} \right\}_{i \in [1,\ell]}$$

and send $(\texttt{rush}, \mathsf{id}^{\mathsf{mul}}, \mathbf{a}, \mathbf{z}_{\mathsf{A}})$ to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and terminate successfully.

Lemma B.1. *In the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$-hybrid Global Random Oracle Model,*

$$\mathsf{REAL}_{\pi^{\ell}_{\mathsf{2PMul}}, \mathcal{A}^{\mathsf{A}}, \mathcal{Z}}(z) \stackrel{\mathsf{c}}{\equiv} \mathsf{IDEAL}_{\mathcal{F}^{\ell}_{\mathsf{2PMul}}, (\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}, \mathcal{A}^{\mathsf{A}}), \mathcal{Z}}(z)$$

for all $z \in \{0,1\}^*$, when $\ell = O(\log(\kappa))$

Proof. Our proof of Lemma B.1 will proceed via a sequence of hybrid experiments, beginning with a real-world execution of the protocol,

$$\mathcal{H}_0 = \left\{ \mathsf{REAL}_{\pi^{\ell}_{\mathsf{2PMul}}, \mathcal{A}^{\mathsf{A}}, \mathcal{Z}}(z) \right\}_{z \in \{0,1\}^*}$$

the outcome of which is characterized Alice's view and by the outputs of Alice and Bob. The information in Alice's view is characterized by the values $\tilde{\mathbf{z}}_{\mathsf{A}}$ and $\hat{\mathbf{z}}_{\mathsf{A}}$ that she receives as output from $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ upon sending it $\tilde{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\alpha}}$, a bit indicating whether Bob aborts, and the value $\boldsymbol{\gamma}_{\mathsf{B}}$ received from Bob when he does not abort. We will argue about the joint distribution of these values.

Our first hybrid will involve the implementation of Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, which tests whether there exists a hypothetical subset of choice bits that would allow Alice to offset her inputs to $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ without detection in a real execution of the protocol, and then aborts based on the probability that exactly those bits are sampled. This step assumes that no more than one such set of choice bits exists. Thus, before we give the hybrid, we will introduce a few lemmas to help us

prove that this is true with overwhelming probability. Specifically, we wish to prove that when $\boldsymbol{\Delta} \leftarrow \mathbb{Z}_q^\ell$ and $\ell = O(\log\log(q))$, there do not exist two distinct bit vectors $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^\ell$ such that $\langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle$ except with negligible probability.

Lemma B.2. *If $\boldsymbol{\Delta} \in \mathbb{Z}_q^\ell$, then there are no more than 3^ℓ unique tuples $(x, \boldsymbol{\beta}, \boldsymbol{\beta}') \in \mathbb{Z}_q \times \{0,1\}^\ell \times \{0,1\}^\ell$ such that*

$$\langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle + x = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle \tag{1}$$

Proof. Consider Equation 1 and notice that at any location i where $\boldsymbol{\beta}_i = \boldsymbol{\beta}'_i$, the contributions of the leftmost term and the right hand side cancel one another. Consequently, all unique solutions can be generated as follows:

1. Partition $\boldsymbol{\Delta}$ into sets $\boldsymbol{\Delta}^{\mathsf{L}}$, $\boldsymbol{\Delta}^{\mathsf{R}}$, and $\boldsymbol{\Delta}^{\mathsf{U}}$.
2. For every possible partition, solve for x in

$$x + \sum_{i \in [1, |\boldsymbol{\Delta}^{\mathsf{L}}|]} \boldsymbol{\Delta}_i^{\mathsf{L}} \;=\; \sum_{i \in [1, |\boldsymbol{\Delta}^{\mathsf{R}}|]} \boldsymbol{\Delta}_i^{\mathsf{R}}$$

The above method exhaustively generates all possible solutions to Equation 1. The total number of possible partitions $(\boldsymbol{\Delta}^{\mathsf{L}}, \boldsymbol{\Delta}^{\mathsf{R}}, \boldsymbol{\Delta}^{\mathsf{U}})$ is 3^ℓ, which proves the lemma. □

Lemma B.3. *Let $\mathbb{Z}_q$ be a κ-bit field, and let $\ell \leq c \cdot \log_2(\kappa)$ for some constant $c > 0$. Then*

$$\Pr_{\boldsymbol{\Delta} \leftarrow \mathbb{Z}_q^\ell} \left[\exists \boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^\ell : \boldsymbol{\beta} \neq \boldsymbol{\beta}' \wedge \langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle \right] \leq \frac{\kappa^{2c}}{2^\kappa}$$

Proof. Let us denote by Expt_i the following experiment:

1. Uniformly sample $\boldsymbol{\Delta} \leftarrow \mathbb{Z}_q^i$.
2. Return 1 if and only if there exists a pair of vectors $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^i$ such that $\boldsymbol{\beta} \neq \boldsymbol{\beta}'$ and $\langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle$.
3. Return 0 otherwise.

In the base case, when $i = 0$, we have

$$\Pr\left[\mathsf{Expt}_1\right] = \Pr_{\Delta \leftarrow \mathbb{Z}_q} \left[\Delta = 0\right] = 1/q$$

In the case that $i > 1$ consider the following: if for a particular assignment of $\boldsymbol{\Delta} \in \mathbb{Z}_q^i$, an assignment of $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^i$ exists to satisfy the above experiment, then when any element is appended to $\boldsymbol{\Delta}$, zeros may may be appended to both $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ in order to satisfy the experiment again. On the other hand, if for a particular assignment of $\boldsymbol{\Delta} \in \mathbb{Z}_q^i$ no satisfying assignment of $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^i$

exists, then when a new element x is appended to $\boldsymbol{\Delta}$ we can only satisfy the experiment by finding $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^i$ such that $\langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle + x = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle$. Thus

$$\begin{aligned}\Pr\left[\mathsf{Expt}_i = 1\right] \leq\ & \Pr\left[\mathsf{Expt}_{i-1} = 1\right] \\ & + \Pr_{\boldsymbol{\Delta} \leftarrow \mathbb{Z}_q^i}\left[\begin{array}{l}\exists \boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^{i-1} : \\ \quad \langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle + \boldsymbol{\Delta}_i = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle\end{array}\right]\end{aligned}$$

where we assume the inner product operator truncates its operands as necessary. By Lemma B.2 it follows that

$$\Pr\left[\mathsf{Expt}_i = 1\right] \leq \Pr\left[\mathsf{Expt}_{i-1} = 1\right] + \frac{3^{i-1}}{q}$$

Thus, we can derive the statement of this lemma,

$$\begin{aligned}& \Pr\left[\mathsf{Expt}_\ell = 1\right] \\ &\leq \sum_{i \in [1,\ell]} \frac{3^{i-1}}{q} = \frac{3^\ell - 1}{2q} \leq \frac{3^{c \cdot \log_2(\kappa)}}{2q} \\ &= \frac{\kappa^{c \cdot \log_2 3}}{2q} \leq \frac{\kappa^{2c}}{2q} \leq \frac{\kappa^{2c}}{2^\kappa}\end{aligned}$$

□

Lemma B.4. *Let $\mathbb{Z}_q$ be a κ-bit field, let $\ell \leq c \cdot \log_2(\kappa)$ for some constant $c > 0$, and let $\boldsymbol{\delta} \in \mathbb{Z}_q^\eta$ be an arbitrary non-zero vector for $\eta = \ell \cdot \xi$ and some positive integer ξ. Then*

$$\Pr_{\boldsymbol{\chi} \leftarrow \mathbb{Z}_q^\ell}\left[\begin{array}{l}\exists j \in [1,\xi],\ \exists \boldsymbol{\beta}, \boldsymbol{\beta}' \in \{0,1\}^\ell : \\ \qquad \boldsymbol{\Delta} = \{\boldsymbol{\delta}_{i \cdot \xi + j} \cdot \boldsymbol{\chi}_i\}_{i \in [1,\ell]} \\ \qquad \implies \boldsymbol{\beta} \neq \boldsymbol{\beta}' \wedge \langle \boldsymbol{\Delta}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\Delta}, \boldsymbol{\beta}' \rangle\end{array}\right] \leq \frac{\xi \cdot \kappa^{2c}}{2^\kappa}$$

Proof. Consider the case where $\xi = 1$. That is, a non-zero $\boldsymbol{\delta} \in \mathbb{Z}_q^\ell$ is fixed, following which a vector $\boldsymbol{\chi} \in \mathbb{Z}_q^\ell$ is sampled uniformly and $\boldsymbol{\Delta}$ is formed by element-wise multiplication of the two. Because $\boldsymbol{\chi}$ is uniform, $\boldsymbol{\Delta}$ must also be, and thus by Lemma B.3 our new lemma holds directly.

Now observe that when $\xi > 1$, this lemma is essentially a statement about the success of any subset of ξ simultaneous repetitions this experiment, under the constraint that they all use the same value of $\boldsymbol{\chi}$. Since $\boldsymbol{\delta}$ is fixed independently of $\boldsymbol{\chi}$, we can simply take the union bound over ξ simultaneous experiments to find that our lemma holds. □

Hybrid $\mathcal{H}_1$. This experiment is the same as $\mathcal{H}_0$, except that Steps 3 through 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ are implemented in order to define $\boldsymbol{\Delta}$, $\boldsymbol{\beta}^*$, and $\mathbf{r}^*$, replacing Bob's instructions in Steps 4 and 6 of $\pi^\ell_{\mathsf{2PMul}}$. This experiment aborts based upon the condition in Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ rather than aborting based on Step 6 of $\pi^\ell_{\mathsf{2PMul}}$. To be clear, this hybrid changes no variables in the view of Alice, and thus it can be distinguished from the real execution only based upon the difference in

the probability of Alice inducing an abort. Thus, we argue that the probability distribution of an abort in this hybrid is negligibly different (statistically) from the distribution in the real world.

Informally stated, Alice may supply the wrong input value to $\mathcal{F}_{\mathsf{COTe}}^{\eta}$ at some locations, which causes a corresponding nonzero value to appear in $\boldsymbol{\Delta}$ in $\mathcal{H}_1$. In the real world she can attempt to compensate for this by adjusting her $\mathbf{r}$ values such that Bob's check in Step 6 of $\pi_{\mathsf{2PMul}}^{\ell}$ passes. Specifically, Alice avoids an abort in the real world when

$$\bigwedge_{j\in[1,\xi]} \left\{ \begin{array}{r} \mathbf{r}_j + \sum_{i\in[0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1}\cdot\tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1}\cdot\hat{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} \\ = \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\mathbf{u}_{i+1} \end{array} \right\} \tag{2}$$

Per $\mathcal{F}_{\mathsf{COTe}}^{\eta}$, we have that for all $j \in [1,\eta]$,

$$\tilde{\mathbf{z}}_{\mathsf{B},j} = \boldsymbol{\beta}_j\cdot\tilde{\boldsymbol{\alpha}}_j - \tilde{\mathbf{z}}_{\mathsf{A},j} \qquad \text{and} \qquad \hat{\mathbf{z}}_{\mathsf{B},j} = \boldsymbol{\beta}_j\cdot\hat{\boldsymbol{\alpha}}_j - \hat{\mathbf{z}}_{\mathsf{A},j}$$

And thus for all $j \in [1,\xi]$, by substitution and per Steps 3 and 4 of $\mathcal{S}_{\mathsf{2PMul}}^{\ell,\mathsf{A}}$,

$$\begin{aligned} &\sum_{i\in[0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1}\cdot\tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1}\cdot\hat{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} \\ =& \sum_{i\in[0,\ell)} \left(\begin{array}{l} \tilde{\boldsymbol{\chi}}_{i+1}\cdot\left(\boldsymbol{\beta}_{i\cdot\xi+j}\cdot\tilde{\boldsymbol{\alpha}}_{i\cdot\xi+j} - \tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j}\right) \\ + \hat{\boldsymbol{\chi}}_{i+1}\cdot\left(\boldsymbol{\beta}_{i\cdot\xi+j}\cdot\hat{\boldsymbol{\alpha}}_{i\cdot\xi+j} - \hat{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j}\right) \end{array} \right) \\ =& \sum_{i\in[0,\ell)} \left(\begin{array}{l} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\left(\tilde{\boldsymbol{\chi}}_{i+1}\cdot\tilde{\boldsymbol{\alpha}}_{i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1}\cdot\hat{\boldsymbol{\alpha}}_{i\cdot\xi+j}\right) \\ - \left(\tilde{\boldsymbol{\chi}}_{i+1}\cdot\tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1}\cdot\hat{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j}\right) \end{array} \right) \\ =& -\mathbf{r}_j^* + \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\left(\boldsymbol{\Delta}_{i\cdot\xi+j} + \mathbf{u}_{i+1}\right) \end{aligned}$$

We then substitute this equality into Equation 2 and find that

$$\begin{aligned} &\bigwedge_{j\in[1,\xi]} \left\{ \begin{array}{r} \sum_{i\in[0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1}\cdot\tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1}\cdot\hat{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} \\ = -\mathbf{r}_j + \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\mathbf{u}_{i+1} \end{array} \right\} \\ =& \bigwedge_{j\in[1,\xi]} \left\{ \begin{array}{r} -\mathbf{r}_j^* + \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\left(\boldsymbol{\Delta}_{i\cdot\xi+j} + \mathbf{u}_{i+1}\right) \\ = -\mathbf{r}_j + \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j}\cdot\mathbf{u}_{i+1} \end{array} \right\} \end{aligned}$$

$$= \bigwedge_{j \in [1,\xi]} \left\{ \sum_{i \in [0,\ell)} \boldsymbol{\beta}_{i \cdot \xi + j} \cdot \boldsymbol{\Delta}_{i \cdot \xi + j} = \mathbf{r}^*_j - \mathbf{r}_j \right\} \tag{3}$$

which is exactly the equation from Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$. If we consider only the subset of indices whereat $\boldsymbol{\Delta}$ is nonzero (the size of which is bounded by ξ), then by Lemma B.4 and the fact that $\ell \leq c \cdot \log(\kappa)$, except with probability $\xi \cdot \kappa^{2c}/2^{\kappa}$ there exists at most one assignment of the corresponding entries in $\boldsymbol{\beta}$ such that this equation is satisfied for a particular combination of $\boldsymbol{\Delta}$, $\mathbf{r}$, and $\mathbf{r}^*_j$. We call this assignment $\boldsymbol{\beta}^*$.

In the case that no such assignment $\boldsymbol{\beta}^*$ exists when $\boldsymbol{\Delta}$, $\mathbf{r}$ and $\mathbf{r}^*_j$ are fixed, Bob will certainly abort in the real world. In this hybrid, Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ checks exhaustively whether such a $\boldsymbol{\beta}^*$ does exist (which requires $2^{\ell} = 2^{c \cdot \log \kappa} = \kappa^c$ steps), and aborts with certainty if not.

In the case that an appropriate assignment $\boldsymbol{\beta}^*$ does exist, Alice avoids an abort in the real world only when Bob chooses $\boldsymbol{\beta} = \boldsymbol{\beta}^*$ (again, considering only the subset of indices whereat $\boldsymbol{\Delta}$ is nonzero). Because Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ chooses $\boldsymbol{\beta}$ from the same distribution as Bob (i.e. uniformly) and aborts under the same condition (i.e. $\boldsymbol{\beta} = \boldsymbol{\beta}^*$), it follows from the existence of exactly one satisfying assignment of $\boldsymbol{\beta}^*$ that Alice sees an abort in $\mathcal{H}_1$ with the same probability as she does in the real protocol. Consequently, this hybrid is distinguishable from the real protocol only when there exists more than one assignment of $\boldsymbol{\beta}^*$ that satisfies Equation 3. This happens with probability $\xi \cdot \kappa^{2c}/2^{\kappa}$.

Hybrid $\mathcal{H}_2$. This experiment is the same as $\mathcal{H}_1$, except that it implements Step 6 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ in order to define $\tilde{\mathbf{a}}$. No variables in view of Alice are changed, but an abort condition is added. Specifically, $\mathcal{H}_2$ aborts if there is not exactly one unique candidate for $\tilde{\mathbf{a}}$, whereas no equivalent behavior exists in $\mathcal{H}_1$. We argue that this event occurs with negligible probability by taking a union bound over the two possible cases: that there is no candidate and that there are too many candidates.

Consider the event in $\mathcal{H}_1$ that there is some $i \in [0, \ell)$ for which there exists no $j \in [1, \xi]$ such that

$$\tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j} = \mathbf{u}_{i+1}$$

Per Step 4 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, this implies that $\boldsymbol{\Delta}_{i \cdot \xi + j} \neq 0$ for the same i and *all* $j \in [1, \xi]$. Since $\mathcal{H}_1$ implemented Step 5 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, the probability that it will not have aborted in this case is less than $2^{-\xi} < 2^{-\kappa}$.

Next, consider the event in $\mathcal{H}_1$ that there are two possible candidates for $\tilde{\mathbf{a}}$. That is, for some $i \in [0, \ell)$ there exist two values $j, j' \in [1, \xi]$ such that

$$\tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} \neq \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j'} \qquad \text{or} \qquad \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j} \neq \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j'}$$

but also such that

$$\tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j} = \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j'} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j'}$$

which implies that

$$\tilde{\boldsymbol{\chi}}_{i+1} \cdot (\tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} - \tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j'}) = \hat{\boldsymbol{\chi}}_{i+1} \cdot (\hat{\boldsymbol{\alpha}}_{i \cdot \xi + j'} - \hat{\boldsymbol{\alpha}}_{i \cdot \xi + j})$$

Note that $\tilde{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\alpha}}$ are fixed, and then $\tilde{\boldsymbol{\chi}}, \hat{\boldsymbol{\chi}}$ are chosen by calling the Random Oracle on the transcript of the protocol thus far, per Step 4 of $\pi^{\ell}_{\mathsf{2PMul}}$. A malicious Alice may attempt to produce different transcripts in order to satisfy this equality, but because there are exactly q satisfying pairs $(\tilde{\boldsymbol{\chi}}_{i+1}, \hat{\boldsymbol{\chi}}_{i+1})$ in the q^2-sized space of all pairs, each of her attempts succeeds with with probability $2^{-\kappa}$. If we call the set of Alice's Random Oracle queries $\mathbf{Q}$, then this event occurs with probability no greater than $|\mathbf{Q}|/2^{\kappa}$, and $\mathcal{H}_2$ is thus computationally indistinguishable from $\mathcal{H}_1$.

We now give two lemmas related to the distribution of Bob's value $\tilde{\mathbf{b}}$ in $\mathcal{H}_2$, which we will use to argue indistinguishability of our next hybrid.

Lemma B.5. *Let $\mathbb{Z}_q$ be a κ-bit field, and $\mu > 0$ an integer. For uniformly sampled $\mathbf{g} \leftarrow \mathbb{Z}_q^{\kappa+\mu}$, $\boldsymbol{\beta} \leftarrow \{0,1\}^{\kappa+\mu}$, and $x \leftarrow \mathbb{Z}_q$, and for all unbounded non-uniform distinguishers $\mathcal{A}$,*

$$\left| \Pr_{\mathbf{g}, \boldsymbol{\beta}} \left[\mathcal{A}\left(\mathbf{g}, \langle \mathbf{g}, \boldsymbol{\beta} \rangle\right) = 1 \right] - \Pr_{\mathbf{g}, x} \left[\mathcal{A}\left(\mathbf{g}, x\right) = 1 \right] \right| \leq 2^{\frac{\mu}{2}}$$

Proof. Follows from a direct application of the Leftover Hash Lemma [ILL89], as shown previously by Impagliazzo and Naor [IN96, Proposition 1.1]. □

Lemma B.6. *In the context of the experiment $\mathcal{H}_2$, for $\mathbf{x} \leftarrow \mathbb{Z}_q^{\ell}$ and for all unbounded non-uniform distinguishers $\mathcal{A}$ playing the role of Alice,*

$$\left| \Pr_{\mathbf{g}, \tilde{\mathbf{b}}} \left[\mathcal{A}\left(\mathbf{g}, \tilde{\mathbf{b}}\right) = 1 \right] - \Pr_{\mathbf{g}, \mathbf{x}} \left[\mathcal{A}(\mathbf{g}, \mathbf{x}) = 1 \right] \right| \leq \ell \cdot 2^{-s}$$

Proof. Recall that in $\mathcal{H}_2$,

$$\tilde{\mathbf{b}} = \left\{ \left\langle \mathbf{g}, \left\{ \boldsymbol{\beta}_{i \cdot \xi + j} \right\}_{j \in [1, \xi]} \right\rangle \right\}_{i \in [0, \ell)}$$

per Step 1 of $\pi^{\ell}_{\mathsf{2PMul}}$, where $\mathbf{g}$ is a uniformly sampled, publicly known vector $\mathbf{g} \leftarrow \mathbb{Z}_q^{\xi}$. Recall also that it is within the power of Alice to set

$$\tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} \neq \tilde{\mathbf{a}}_{i+1} \implies \boldsymbol{\Delta}_{i \cdot \xi + j} \neq 0$$

for any subset of $i \in [0, \ell), j \in [1, \xi]$, and that per Step 5 of $\mathcal{S}^{\ell, \mathsf{A}}_{\mathsf{2PMul}}$, $\mathcal{H}_2$ will have aborted with probability

$$2^{-\sum_{i \in [1, \eta]} \lceil \boldsymbol{\Delta}_i / q \rceil}$$

If $\mathcal{H}_2$ has not already aborted, then we must assume that Alice knows the value of $\boldsymbol{\beta}_i$ for all $i \in [1, \eta]$ such that $\boldsymbol{\Delta}_i \neq 0$. Given this additional information, and

accounting for the probability that acquiring this information does not induce an abort, the advantage that any unbounded distinguisher $\mathcal{A}$ has in distinguishing $\tilde{\mathbf{b}}$ from a set sampled uniformly from from $\mathbb{Z}_q^\ell$ is given by

$$\mathsf{Adv}_{\mathcal{A}} = \left| \begin{array}{r} \Pr\left[\mathcal{A}\left(\mathbf{g}, \left\{\sum_{\substack{j\in[1,\xi]:\\ \boldsymbol{\Delta}_{i\cdot\xi+j}\neq 0}} \mathbf{g}_j \cdot \boldsymbol{\beta}_{i\cdot\xi+j}\right\}_{i\in[0,\ell)}\right) = 1\right] \\ - \Pr\left[\mathcal{A}\left(\mathbf{g}, \mathbf{x} \leftarrow \mathbb{Z}_q^\ell\right) = 1\right] \end{array} \right| \cdot 2^{-\sum_{i\in[1,\eta]} \lceil \boldsymbol{\Delta}_i/q \rceil}$$

Observe that this is essentially a statement about ℓ independent experiments (each with a distinguisher $\mathcal{A}'$), over which we can instead take a union bound to find that the distinguisher $\mathcal{A}$ has an advantage no greater than

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A}} &\leq 2^{-\sum_{i\in[1,\eta]} \lceil \boldsymbol{\Delta}_i/q \rceil} \cdot \sum_{i\in[0,\ell)} \mathsf{Adv}_{\mathcal{A}'} \\ &= 2^{-\sum_{i\in[1,\eta]} \lceil \boldsymbol{\Delta}_i/q \rceil} \cdot \sum_{i\in[0,\ell)} \left| \begin{array}{r} \Pr\left[\mathcal{A}'\left(\mathbf{g}, \sum_{\substack{j\in[1,\xi]:\\ \boldsymbol{\Delta}_{i\cdot\xi+j}\neq 0}} \mathbf{g}_j \cdot \boldsymbol{\beta}_{i\cdot\xi+j}\right) = 1\right] \\ - \Pr\left[\mathcal{A}'\left(\mathbf{g}, x \leftarrow \mathbb{Z}_q\right) = 1\right] \end{array} \right| \end{aligned}$$

Via Lemma B.5 and the fact that $\xi = \kappa + 2s$, the advantage of the distinguisher $\mathcal{A}'$ in each experiment i for $i \in [1, \ell]$ is at most

$$\mathsf{Adv}_{\mathcal{A}'} \leq 2^{\sum_{j\in[1,\xi]} \lceil \boldsymbol{\Delta}_{i\cdot\xi+j}/q \rceil/2 - s}$$

which gives $\mathcal{A}$ a maximum total advantage of

$$\mathsf{Adv}_{\mathcal{A}} \leq 2^{-\sum_{i\in[1,\eta]} \lceil \boldsymbol{\Delta}_i/q \rceil} \cdot \sum_{i\in[0,\ell)} 2^{\sum_{j\in[1,\xi]} \lceil \boldsymbol{\Delta}_{i\cdot\xi+j}/q \rceil/2 - s} \leq \ell \cdot 2^{-s} \qquad \square$$

Hybrid $\mathcal{H}_3$. This experiment is the same as $\mathcal{H}_2$, except that it implements Steps 1, 2, and 7 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, replacing Bob's instructions in Steps 3, 7, and 8 of $\pi^{\ell}_{\mathsf{2PMul}}$. This hybrid differs from $\mathcal{H}_2$ in the way that Bob's output $\mathbf{z}_\mathsf{B}$ is calculated. Specifically, in this hybrid $\mathbf{z}_\mathsf{B}$ is calculated by $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. Note that this value is not in Alice's view, but nevertheless influences the output of the experiment. This hybrid also differs with respect to the distribution of $\boldsymbol{\gamma}_\mathsf{B}$. First, we will discuss the distribution of $\mathbf{z}_\mathsf{B}$.

In $\mathcal{H}_3$, Bob's output $\mathbf{z}_\mathsf{B}$ is received from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, and thus

$$\mathbf{z}_\mathsf{B} = \left\{\mathbf{a}_i \cdot \mathbf{b}_i - \mathbf{z}_{\mathsf{A},i}\right\}_{i\in[1,\ell]} \tag{4}$$

while Alice's output $\mathbf{z}_{\mathsf{A}}$ is chosen uniformly by $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$. Per Step 7 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, Alice's input $\mathbf{a}$ is extracted as

$$\mathbf{a} = \left\{\tilde{\mathbf{a}}_i + \boldsymbol{\gamma}_{\mathsf{A},i}\right\}_{i\in[1,\ell]} \tag{5}$$

and per the same step,

$$\mathbf{z}_{\mathsf{A}} = \left\{\mathbf{a}_i \cdot \boldsymbol{\gamma}_{\mathsf{B},i} - \mathbf{d}_i + \sum_{j\in[1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},(i-1)\cdot\xi+j}\right\}_{i\in[1,\ell]} \tag{6}$$

Consequently, by sequential substitution of Equations 5 and 6 into Equation 4, Bob's output is calculated in $\mathcal{H}_3$ as

$$\mathbf{z}_{\mathsf{B}} = \left\{\begin{array}{l}\left(\mathbf{b}_{i+1} - \boldsymbol{\gamma}_{\mathsf{B},i+1}\right) \cdot \left(\tilde{\mathbf{a}}_{i+1} + \boldsymbol{\gamma}_{\mathsf{A},i+1}\right) \\ + \mathbf{d}_{i+1} - \sum_{j\in[1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j}\end{array}\right\}_{i\in[0,\ell)} \tag{7}$$

In $\mathcal{H}_2$ on the other hand, Step 8 of $\pi^{\ell}_{\mathsf{2PMul}}$ gives us

$$\mathbf{z}_{\mathsf{B}} = \left\{\tilde{\mathbf{b}}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{A},i+1} + \sum_{j\in[1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j}\right\}_{i\in[0,\ell)} \tag{8}$$

Recall that per the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ specification and Step 3 of $\pi^{\ell}_{\mathsf{2PMul}}$,

$$\tilde{\mathbf{z}}_{\mathsf{B}} = \left\{\tilde{\boldsymbol{\alpha}}_i \cdot \boldsymbol{\beta}_i - \tilde{\mathbf{z}}_{\mathsf{A},i}\right\}_{i\in[1,\eta]} \tag{9}$$

and that Step 6 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$ defined

$$\mathbf{d} = \left\{\sum_{j\in[1,\xi]} \mathbf{g}_j \cdot \boldsymbol{\beta}^*_{i\cdot\xi+j} \cdot \left(\tilde{\boldsymbol{\alpha}}_{i\cdot\xi+j} - \tilde{\mathbf{a}}_{i+1}\right)\right\}_{i\in[0,\ell)} \tag{10}$$

and that per Steps 5 and 6 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$,

$$\forall i \in [0,\ell)\, \forall j \in [1,\xi],\ \tilde{\boldsymbol{\alpha}}_{i\cdot\xi+j} \neq \tilde{\mathbf{a}}_{i+1} \implies \boldsymbol{\beta}_{i\cdot\xi+j} = \boldsymbol{\beta}^*_{i\cdot\xi+j}$$

and finally recall that per Steps 1 and 7 of $\pi^{\ell}_{\mathsf{2PMul}}$ respectively,

$$\tilde{\mathbf{b}} = \left\{\left\langle \mathbf{g}, \left\{\boldsymbol{\beta}_{i\cdot\xi+j}\right\}_{j\in[1,\xi]}\right\rangle\right\}_{i\in[0,\ell)} \tag{11}$$

$$\tilde{\mathbf{b}} = \left\{\mathbf{b}_i - \boldsymbol{\gamma}_{\mathsf{B},i}\right\}_{i\in[1,\ell]} \tag{12}$$

Consequently, by sequential substitution of Equations 9, 10, 11, and 12 into Equation 8, we have in $\mathcal{H}_2$

$$\begin{aligned}
\mathbf{z}_{\mathsf{B}} &= \left\{ \begin{array}{l} \tilde{\mathbf{b}}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{A},i+1} \\ + \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \left(\tilde{\boldsymbol{\alpha}}_{i \cdot \xi + j} \cdot \boldsymbol{\beta}_{i \cdot \xi + j} - \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \right) \end{array} \right\}_{i \in [0,\ell)} \\
&= \left\{ \begin{array}{l} \tilde{\mathbf{b}}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{A},i+1} + \mathbf{d}_{i+1} \\ + \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \left(\tilde{\mathbf{a}}_{i+1} \cdot \boldsymbol{\beta}_{i \cdot \xi + j} - \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \right) \end{array} \right\}_{i \in [0,\ell)} \\
&= \left\{ \begin{array}{l} \tilde{\mathbf{b}}_{i+1} \cdot \left(\tilde{\mathbf{a}}_{i+1} + \boldsymbol{\gamma}_{\mathsf{A},i+1} \right) \\ + \mathbf{d}_{i+1} - \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \end{array} \right\}_{i \in [0,\ell)} \\
&= \left\{ \begin{array}{l} \left(\mathbf{b}_{i+1} - \boldsymbol{\gamma}_{\mathsf{B},i+1} \right) \cdot \left(\tilde{\mathbf{a}}_{i+1} + \boldsymbol{\gamma}_{\mathsf{A},i+1} \right) \\ + \mathbf{d}_{i+1} - \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i \cdot \xi + j} \end{array} \right\}_{i \in [0,\ell)}
\end{aligned}$$

which exactly the same as Equation 7. Thus, this change is purely syntactical, and the distinguishability of $\mathcal{H}_3$ and $\mathcal{H}_2$ is conditioned solely upon the distinguishability of the distributions of $\boldsymbol{\gamma}_{\mathsf{B}}$ in each.

The distribution of $\boldsymbol{\gamma}_{\mathsf{B}}$ in $\mathcal{H}_3$ varies based upon the cases in Step 7 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$. If the second branch of Step 7 is taken, then we have $\boldsymbol{\gamma}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\ell}$, and the distribution of $\boldsymbol{\gamma}_{\mathsf{B}}$ is uniform. If the first branch of Step 7 is taken, we have for every $i \in [1,\ell]$ either that $\boldsymbol{\gamma}_{\mathsf{B},i} \leftarrow \mathbb{Z}_q$, or

$$\boldsymbol{\gamma}_{\mathsf{B},i} = \frac{\mathbf{z}_{\mathsf{A},i} + \mathbf{d}_i - \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},(i-1) \cdot \xi + j}}{\mathbf{a}_i}$$

whence we derive, via substitution of Equation 4

$$\boldsymbol{\gamma}_{\mathsf{B},i} = \mathbf{b}_i + \frac{\mathbf{d}_i - \mathbf{z}_{\mathsf{B},i} - \sum\limits_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},(i-1) \cdot \xi + j}}{\mathbf{a}_i}$$

where $\mathbf{z}_{\mathsf{B},i}$ is sampled uniformly by $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. As a consequence, the distribution of $\boldsymbol{\gamma}_{\mathsf{B}}$ is uniform regardless of which branch is taken in Step 7 of $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$.

In $\mathcal{H}_2$, on the other hand, we have

$$\boldsymbol{\gamma}_{\mathsf{B}} = \left\{ \mathbf{b}_i - \tilde{\mathbf{b}}_i \right\}_{i \in [1,\ell]}$$

per Step 7 of $\pi^{\ell}_{\mathsf{2PMul}}$. We will assume that $\mathbf{b}$ is completely known to the distinguisher, and thus $\mathcal{H}_3$ is indistinguishable from $\mathcal{H}_2$ if and only if $\tilde{\mathbf{b}}$ is

indistinguishable from uniform. By Lemma B.6, we have that these distributions are distinguishable with probability at most $\ell \cdot 2^{-s}$. Since $\ell = c \cdot \log_2(\kappa)$ for some constant c, it follows that an unbounded adversary can distinguish $\mathcal{H}_3$ from $\mathcal{H}_2$ with probability at most $c \cdot \log_2(\kappa) \cdot 2^{-s}$. Because we have now implemented all steps in $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, we have

$$\mathcal{H}_3 = \left\{ \mathsf{IDEAL}_{\mathcal{F}^{\ell}_{\mathsf{2PMul}},(\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}},\mathcal{A}^{\mathsf{A}}),\mathcal{Z}}(z) \right\}_{z \in \{0,1\}^*}$$

and by transitivity we have $\mathcal{H}_3 \overset{\mathsf{c}}{\equiv} \mathcal{H}_0$, proving Lemma B.1. □

B.2 Simulating Against Bob

Simulator 2. Two-party Multiplication against Bob $\left(\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}\right)$:

This simulator interposes between the ideal functionality $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and a malicious Bob running the $\pi^{\ell}_{\mathsf{2PMul}}$ protocol. It is parameterized by the statistical security parameter s and the symmetric security parameter κ, with $\xi = \kappa + 2s$ and $\eta = \xi \cdot \ell$. It also makes use of a gadget vector $\mathbf{g}$ of the same form as that used by $\pi^{\ell}_{\mathsf{2PMul}}$. It plays the role of the functionality $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ in its interaction with Bob, and it can observe Alice's queries to the random oracle H.

Init: Receive message (`init`) from Bob on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, and then send (`init`) to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. Upon receipt of (`init-complete`) from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, send (`init-complete`) to Bob on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$.

Multiplication:

1. On receiving $(\mathtt{choose}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\beta})$ from Bob on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, if $\mathsf{id}^{\mathsf{ext}}$ is a fresh index, then compute

$$\tilde{\mathbf{b}} := \left\{ \left\langle \mathbf{g}, \left\{ \boldsymbol{\beta}_{i \cdot \xi + j} \right\}_{j \in [1,\xi]} \right\rangle \right\}_{i \in [0,\ell)}$$

   and send $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, where $\mathsf{id}^{\mathsf{mul}}$ is a fresh index.

2. Wait for $(\mathtt{alice\text{-}ready}, \mathsf{id}^{\mathsf{mul}})$ from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. Upon receipt, sample

$$\tilde{\mathbf{z}}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\eta} \qquad \text{and} \qquad \hat{\mathbf{z}}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\eta}$$

   and then compute $\boldsymbol{\omega}_{\mathsf{B}}$ as

$$\boldsymbol{\omega}_{\mathsf{B}} := \left\{ \tilde{\mathbf{z}}_{\mathsf{B},j} \| \hat{\mathbf{z}}_{\mathsf{B},j} \right\}_{j \in [1,\eta]}$$

   and send $(\mathtt{correlation}, \mathsf{id}^{\mathsf{mul}}, \boldsymbol{\omega}_{\mathsf{B}})$ to Bob on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$.

3. Engage in the coin tossing protocol (corresponding to Step 4 of $\pi^{\ell}_{\mathsf{2PMul}}$) with Bob to compute $\tilde{\boldsymbol{\chi}}$ and $\hat{\boldsymbol{\chi}}$. Sample $\mathbf{u} \leftarrow \mathbb{Z}_q^{\ell}$, and compute

$$\mathbf{r} := \left\{ \sum_{i \in [0,\ell)} \begin{pmatrix} \boldsymbol{\beta}_{i \cdot \xi + j} \cdot \mathbf{u}_{i+1} \\ - \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\mathbf{z}}_{\mathsf{B}, i \cdot \xi + j} \\ - \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\mathbf{z}}_{\mathsf{B}, i \cdot \xi + j} \end{pmatrix} \right\}_{j \in [1,\xi]}$$

and send $\mathbf{r}$ and $\mathbf{u}$ to Bob on behalf of Alice.

4. Wait for a single message each from both $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and Bob. Regardless of the sequence in which the two messages arrive, perform the following steps:

   - When $\boldsymbol{\gamma}_{\mathsf{B}}$ is received from Bob, compute

   $$\mathbf{b} := \left\{ \boldsymbol{\gamma}_{\mathsf{B},i} + \tilde{\mathbf{b}}_i \right\}_{i \in [1,\ell]}$$

   and send $(\mathtt{input}, \mathsf{id}^{\mathsf{mul}}, \mathbf{b})$ to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$.

   - When $(\mathtt{output}, \mathsf{id}^{\mathsf{mul}}, \mathbf{z}_{\mathsf{B}})$ is received from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, define $\boldsymbol{\gamma}_{\mathsf{A}} \in \mathbb{Z}_q^{\ell}$ such that for $i \in [1,\ell]$, if $\tilde{\mathbf{b}}_i = 0$ then $\boldsymbol{\gamma}_{\mathsf{A},i} \leftarrow \mathbb{Z}_q$, and otherwise

   $$\boldsymbol{\gamma}_{\mathsf{A},i} := \frac{\mathbf{z}_{\mathsf{B},i} - \sum_{j \in [1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{B},(i-1)\cdot\xi+j}}{\tilde{\mathbf{b}}_i}$$

   and send $\boldsymbol{\gamma}_{\mathsf{A}}$ to Bob on behalf of Alice.

   When both messages have been received and the above steps are complete, terminate successfully.

Lemma B.7. *In the $\mathcal{F}^{\eta}_{\mathsf{COTe}}$-hybrid Global Random Oracle Model, for $z \in \{0,1\}^*$*

$$\mathsf{REAL}_{\pi^{\ell}_{\mathsf{2PMul}}, \mathcal{A}^{\mathsf{B}}, \mathcal{Z}}(z) = \mathsf{IDEAL}_{\mathcal{F}^{\ell}_{\mathsf{2PMul}}, (\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}, \mathcal{A}^{\mathsf{B}}), \mathcal{Z}}(z)$$

Proof. The outputs of the experiments are characterized by Bob's view and by the values $\mathbf{z}_{\mathsf{A}}$ and $\mathbf{z}_{\mathsf{B}}$. The information in Bob's view is characterized by the outputs $\tilde{\mathbf{z}}_{\mathsf{B}}$ and $\hat{\mathbf{z}}_{\mathsf{B}}$ that he receives from $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, and by the messages $\mathbf{u}$, $\mathbf{r}$, and $\boldsymbol{\gamma}_{\mathsf{A}}$ that he receives from Alice. Thus, we will argue that the joint distribution of all these values is identical in the real and ideal worlds.

In the real protocol, per Step 3 of $\pi^{\ell}_{\mathsf{2PMul}}$, $\tilde{\mathbf{z}}_{\mathsf{B}}$ and $\hat{\mathbf{z}}_{\mathsf{B}}$ are chosen uniformly by $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, subject to the relationships

$$\tilde{\mathbf{z}}_{\mathsf{A},i} + \tilde{\mathbf{z}}_{\mathsf{B},i} = \boldsymbol{\beta}_i \cdot \tilde{\mathbf{a}}_{\lceil i/\xi \rceil} \qquad \text{and} \qquad \hat{\mathbf{z}}_{\mathsf{A},i} + \hat{\mathbf{z}}_{\mathsf{B},i} = \boldsymbol{\beta}_i \cdot \hat{\mathbf{a}}_{\lceil i/\xi \rceil}$$

for $i \in [1,\eta]$. In the simulation, on the other hand, $\tilde{\mathbf{z}}_{\mathsf{B}}$ and $\hat{\mathbf{z}}_{\mathsf{B}}$ are uniformly sampled per Step 2 of $\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}$, and thus are identically distributed to their real-world counterparts from Bob's perspective.

In the real protocol, per Step 7 of $\pi^{\ell}_{\mathsf{2PMul}}$,

$$\boldsymbol{\gamma}_{\mathsf{A}} = \{\mathbf{a}_i - \tilde{\mathbf{a}}_i\}_{i\in[1,\ell]}$$

Since $\tilde{\mathbf{a}}$ is sampled uniformly per Step 2 of $\pi^{\ell}_{\mathsf{2PMul}}$ (and no trace of it is in Bob's view), $\boldsymbol{\gamma}_{\mathsf{A}}$ is uniform from Bob's perspective. On the other hand, the in the ideal experiment per Step 4 of $\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}$, the simulator computes $\boldsymbol{\gamma}_{\mathsf{A}}$ to satisfy

$$\boldsymbol{\gamma}_{\mathsf{A}} := \left\{ \frac{\mathbf{z}_{\mathsf{B},i+1} - \sum\limits_{j\in[1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j}}{\tilde{\mathbf{b}}_{i+1}} \right\}_{i\in[0,\ell)}$$

where $\mathbf{z}_{\mathsf{B}}$ is uniformly chosen by $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and not (yet) known to Bob, which implies that $\boldsymbol{\gamma}_{\mathsf{A}}$ is distributed uniformly from his perspective. Note that in the edge case where $\tilde{\mathbf{b}}_i = 0$ for some $i \in [1,\ell]$, any uniformly chosen $\boldsymbol{\gamma}_{\mathsf{A},i}$ will produce a correct output.

In the real protocol, $\hat{\mathbf{a}}$ is sampled uniformly per Step 2 of $\pi^{\ell}_{\mathsf{2PMul}}$ and does not occur anywhere else in Bob's view; thus

$$\mathbf{u} = \{\tilde{\boldsymbol{\chi}}_i \cdot \tilde{\mathbf{a}}_i + \hat{\boldsymbol{\chi}}_i \cdot \hat{\mathbf{a}}_i\}_{i\in[1,\ell]}$$

(where $\tilde{\boldsymbol{\chi}}$ and $\hat{\boldsymbol{\chi}}$ are non-zero values known to Bob) is uniform from Bob's perspective. In the simulation $\mathbf{u}$ is sampled uniformly by $\mathcal{S}^{\ell,\mathsf{A}}_{\mathsf{2PMul}}$, and thus is identically distributed.

We have in the real experiment per Step 6 of $\pi^{\ell}_{\mathsf{2PMul}}$ that for $j \in [1,\xi]$, $\mathbf{r}$ is consistent with

$$\mathbf{r}_j + \sum_{i\in[0,\ell)} \tilde{\boldsymbol{\chi}}_{i+1} \cdot \tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} + \hat{\boldsymbol{\chi}}_{i+1} \cdot \hat{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j} = \sum_{i\in[0,\ell)} \boldsymbol{\beta}_{i\cdot\xi+j} \cdot \mathbf{u}_{i+1}$$

In the simulation, $\boldsymbol{\beta}$ is received from Bob on behalf of $\mathcal{F}^{\eta}_{\mathsf{COTe}}$, the values $\mathbf{u}$, $\tilde{\mathbf{z}}_{\mathsf{B}}$, and $\hat{\mathbf{z}}_{\mathsf{B}}$ are chosen by the simulator, and $\tilde{\boldsymbol{\chi}}$ and $\hat{\boldsymbol{\chi}}$ are public. In Step 3, $\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}$ can solve for $\mathbf{r}$, maintaining the correct distribution.

Finally, we must consider the distribution of the output $\mathbf{z}_{\mathsf{A}}$, defined in the real world per Step 8 of $\pi^{\ell}_{\mathsf{2PMul}}$ by

$$\mathbf{z}_{\mathsf{A}} = \left\{ \mathbf{a}_{i+1} \cdot \boldsymbol{\gamma}_{\mathsf{B},i+1} + \sum_{j\in[1,\xi]} \mathbf{g}_j \cdot \tilde{\mathbf{z}}_{\mathsf{A},i\cdot\xi+j} \right\}_{i\in[0,\ell)} \tag{13}$$

Recall that per $\mathcal{F}^{\eta}_{\mathsf{COTe}}$ and Step 3 of $\pi^{\ell}_{\mathsf{2PMul}}$,

$$\tilde{\mathbf{z}}_{\mathsf{A}} = \left\{ \tilde{\mathbf{a}}_{\lceil i/\xi \rceil} \cdot \boldsymbol{\beta}_i - \tilde{\mathbf{z}}_{\mathsf{B},i} \right\}_{i\in[1,\eta]} \tag{14}$$

and recall that

$$\tilde{\mathbf{b}} := \left\{ \left\langle \mathbf{g}, \left\{ \boldsymbol{\beta}_{i\cdot\xi+j} \right\}_{j\in[1,\xi]} \right\rangle \right\}_{i\in[0,\ell)} \tag{15}$$

and that per Step 7 of $\pi^{\ell}_{\mathsf{2PMul}}$,

$$\tilde{\mathbf{a}} = \left\{\mathbf{a}_i - \boldsymbol{\gamma}_{\mathsf{A},i}\right\}_{i\in[1,\ell]} \tag{16}$$

and Step 4 of $\mathcal{S}^{\ell,\mathsf{B}}_{\mathsf{2PMul}}$

$$\mathbf{b} = \left\{\tilde{\mathbf{b}}_i + \boldsymbol{\gamma}_{\mathsf{B},i}\right\}_{i\in[1,\ell]} \tag{17}$$

and that Step 8 of $\pi^{\ell}_{\mathsf{2PMul}}$ defines

$$\mathbf{z}_{\mathsf{B}} = \left\{\tilde{\mathbf{b}}_{i+1}\cdot\boldsymbol{\gamma}_{\mathsf{A},i+1} + \sum_{j\in[1,\xi]}\mathbf{g}_j\cdot\tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j}\right\}_{i\in[0,\ell)} \tag{18}$$

Thus sequential substitution of Equations 14, 16, 15, 18, and 17 into 13 yields

$$\begin{aligned}
\mathbf{z}_{\mathsf{A}} &= \left\{\begin{array}{l}\mathbf{a}_{i+1}\cdot\boldsymbol{\gamma}_{\mathsf{B},i+1}\\ +\sum_{j\in[1,\xi]}\mathbf{g}_j\cdot\left(\tilde{\mathbf{a}}_{i+1}\cdot\boldsymbol{\beta}_{i\cdot\xi+j} - \tilde{\mathbf{z}}_{\mathsf{B},i\cdot\xi+j}\right)\end{array}\right\}_{i\in[0,\ell)}\\
&= \left\{\begin{array}{l}\mathbf{a}_i\cdot\boldsymbol{\gamma}_{\mathsf{B},i} + \tilde{\mathbf{a}}_i\cdot\tilde{\mathbf{b}}_i\\ -\sum_{j\in[1,\xi]}\mathbf{g}_j\cdot\tilde{\mathbf{z}}_{\mathsf{B},(i-1)\cdot\xi+j}\end{array}\right\}_{i\in[1,\ell]}\\
&= \left\{\begin{array}{l}\mathbf{a}_i\cdot\boldsymbol{\gamma}_{\mathsf{B},i} + \mathbf{a}_i\cdot\tilde{\mathbf{b}}_i - \boldsymbol{\gamma}_{\mathsf{A},i}\cdot\tilde{\mathbf{b}}_i\\ -\sum_{j\in[1,\xi]}\mathbf{g}_j\cdot\tilde{\mathbf{z}}_{\mathsf{B},(i-1)\cdot\xi+j}\end{array}\right\}_{i\in[1,\ell]}\\
&= \left\{\mathbf{a}_i\cdot\boldsymbol{\gamma}_{\mathsf{B},i} + \mathbf{a}_i\cdot\tilde{\mathbf{b}}_i - \mathbf{z}_{\mathsf{B},i}\right\}_{i\in[1,\ell]}\\
&= \left\{\mathbf{a}_i\cdot\mathbf{b}_i - \mathbf{z}_{\mathsf{B},i}\right\}_{i\in[1,\ell]}
\end{aligned}$$

which is identical to the relation maintained by $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ in the ideal execution. The views produced by real and simulated executions of $\pi^{\ell}_{\mathsf{2PMul}}$ are therefore distributed identically to an adversary corrupting Bob. □

C Proof of Security for t-Party Inverse Sampling

Theorem 4.1. *The protocol $\pi^{n,t}_{\mathsf{Inv}}$ UC-realizes the functionality $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ for a κ-bit elliptic curve group $(\mathbb{G}, G, q)$ in the $(\mathcal{F}^{\ell}_{\mathsf{2PMul}}, \mathcal{F}^{n}_{\mathsf{Com}})$-hybrid model, in the presence of a computationally unbounded malicious adversary statically corrupting up to $t-1$ parties.*

Proof. After the init phase, the online phases of the $\pi^{n,t}_{\mathsf{Inv}}$ protocol are run with only t parties who are indexed by $\mathbf{P}$. We make the simplifying assumption that the adversary makes complete use of its power by corrupting $t-1$ parties with indices given by $\mathbf{P}^*$, such that $\mathbf{P}^* \subset \mathbf{P}$, and only a single honest party with index h remains to participate in the protocol. It should be remembered that $\mathbf{P}$ and $\mathbf{P}^*$ need not have any particular relationship in general, and may even be disjoint, but the adversary in any other case is strictly weaker than the adversary we assume here. Our simulator $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$ for the protocol $\pi^{n,t}_{\mathsf{Inv}}$ extracts the inputs and characteristic values of the corrupt parties jointly. We notate these joint values with asterisks in place of the party-index subscript; so, for example,

$$u_* = \sum_{i \in \mathbf{P}^*} u_i$$

Before we give a formal description of our simulator and hybrids, we outline our argument informally. The adversary's view is characterized by the values R_h, Γ^1_h, and ϕ_h along with the pattern of aborts. Because the simulator plays the role of $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, it can observe the adversary's inputs and outputs in the Inverse Sampling phase of $\pi^{n,t}_{\mathsf{Inv}}$ and use this information to choose one of three simulation strategies for the subsequent Consistency Check phase. If the adversary's behavior is consistent with an honest execution of $\pi^{n,t}_{\mathsf{Inv}}$, then the simulator queries $\mathcal{F}^{n,t}_{\mathsf{Inv}}$, supplying an extracted value of u_* and receiving R and v_* in return; because the adversary has behaved honestly, Γ^1_h and ϕ_h can be calculated as a function of these values and the corrupted parties' outputs from $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. This is the first strategy, and in this case, the simulation is perfect.

The second and third strategies are used respectively in the case that the adversary to sets $k = 0$ (and thus attempts to force the protocol to invert zero), and the case that the adversary behaves dishonestly by using inconsistent values within the Inverse Sampling phase or between this phase and the Consistency Check. Both strategies are similar in that the simulator avoids querying $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ and instead effectively runs the code of an honest party before aborting unconditionally. Both strategies are distinguishable from the real world only if the adversary can behave dishonestly and yet avoid an abort in the real world. We argue in both cases that the probability of such an event is negligible.

Regardless of which simulation strategy is taken, this simulator can be generalized to the case of acting on behalf of multiple honest parties via the same transformation: for each additional honest party, the simulator runs an instance of the code in $\pi^{n,t}_{\mathsf{Inv}}$ that interacts with the corrupt parties, and for the purpose of simulating party h, these virtual parties are treated adversarially.

Simulator 3. *t*-party Modular Inverse Sampling against $\mathbf{P}^*$ $\left(\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}\right)$:

This simulator interposes between an evaluation of the $\pi^{n,t}_{\mathsf{Inv}}$ protocol involving the group of $t-1$ malicious parties indexed by $\mathbf{P}^*$ and the ideal functionality $\mathcal{F}^{n,t}_{\mathsf{Inv}}$. In the online stages of the protocol, it acts on behalf of the remaining honest party $\mathcal{P}_h$ with index h, and plays the roles the functionalities $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, $\mathcal{F}^{n}_{\mathsf{Com}}$, and $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$ in their interactions with the corrupted parties.

Init: Receive message $(\texttt{init})$ from all parties in $\mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ and then send $(\texttt{init})$ to $\mathcal{F}_{\mathsf{Inv}}^{n,t}$.

Inverse Sampling:

1. Send $(\texttt{committed}, \mathsf{id}_{h,1}^{\mathsf{com}}, h)$ to every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}^{n}$, using a fresh value for $\mathsf{id}_{h,1}^{\mathsf{com}}$. Wait until every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ has sent $(\texttt{commit}, \mathsf{id}_{i,1}^{\mathsf{com}}, \phi_i)$ to $\mathcal{F}_{\mathsf{Com}}^{n}$. Compute

$$\phi_* := \prod_{i \in \mathbf{P}^*} \phi_i$$

2. Let $\ell = 2$. For each $i \in \mathbf{P}^*$:
   (a) If $i < j$ then Send $(\texttt{bob-ready}, \mathsf{id}_{i,h}^{\mathsf{mul}})$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$
   (b) Otherwise wait for $(\texttt{preprocess}, \mathsf{id}_{h,i}^{\mathsf{mul}})$ (on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$) from $\mathcal{P}_i$ and respond with $(\texttt{alice-ready}, \mathsf{id}_{i,h}^{\mathsf{mul}})$

3. For $\rho \in [1, \lceil \log_2(t) \rceil]$:
   (a) Let $\mathbf{P}^{\rho,h} \subset \mathbf{P}$ be the vector of indices of the parties that interact with $\mathcal{P}_h$ in iteration ρ. For all $i \in \mathbf{P}^{\rho,j} \cap \mathbf{P}^*$, interact with $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ as appropriate in order to receive $\boldsymbol{\psi}_i^{\rho}$, $\mathcal{P}_i$'s input in its multiplication with $\mathcal{P}_h$ in iteration ρ.
      - If $\mathcal{P}_i$ is playing the role of Alice and submitted her input via the $\texttt{Alice-input-rush}$ interface of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, then she also submitted $\boldsymbol{\zeta}_i^{\rho,h}$, and no response is necessary
      - Otherwise, sample $\boldsymbol{\zeta}_i^{\rho,h} \leftarrow \mathbb{Z}_q^{\ell}$ and send $(\texttt{output}, \mathsf{id}_{i,h}^{\mathsf{mul}}, \boldsymbol{\zeta}_i^{\rho,h})$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$.

4. Compute the effective joint inputs of the corrupted parties given the interactions in Step 3.

$$\boldsymbol{\psi}_* := \left\{ \prod_{\rho \in [1, \lceil \log_2(t) \rceil]} \sum_{i \in \mathbf{P}^{\rho,h}} \boldsymbol{\psi}_{i,l}^{\rho} \right\}_{l \in [1,\ell]}$$

   and then let

$$k_* := \boldsymbol{\psi}_{*,1} \qquad \text{and} \qquad e^k := \boldsymbol{\psi}_{*,2} - \frac{\phi_*}{k_*}$$

5. Compute the expected joint product shares $u_*, \tilde{v}_*$ for the corrupt parties, given the interactions in Step 3. As before, let $\mathbf{P}^{\rho,h} \subset \mathbf{P}$ be the vector of indices of the parties that interact with $\mathcal{P}_h$ in iteration ρ, and now let $\mathbf{P}^{\lceil \log_2(t) \rceil + 1, h} = \mathbf{P}^*$ (for notational convenience; does not correspond to

an actual interaction). Then, for $\mathcal{P}_i$ we have the vector $\boldsymbol{\zeta}_i^\rho \in \mathbb{Z}_q^\ell$ defined recursively for all $l \in [1, \ell]$ by

$$\boldsymbol{\zeta}_{i,l}^\rho := \begin{cases} \boldsymbol{\zeta}_{i,l}^{\rho,h} + \boldsymbol{\psi}_{i,l}^\rho \cdot \sum_{\substack{j \in \bigcup_{\rho' \in [1,\rho-1]} \mathbf{P}^{\rho',h}}} \boldsymbol{\zeta}_{j,l}^{\rho-1} & \text{if } i \in \mathbf{P}^{\rho,h} \\ \boldsymbol{\zeta}_{i,l}^{\rho-1} \cdot \sum_{j \in \mathbf{P}^{\rho,h}} \boldsymbol{\psi}_{j,l}^\rho & \text{if } i \notin \mathbf{P}^{\rho,h} \end{cases}$$

and finally we can compute the joint output shares for the corrupt parties

$$u_* := \sum_{i \in \mathbf{P}^*} \boldsymbol{\zeta}_{i,1}^{\lceil \log_2(t) \rceil} \qquad \text{and} \qquad \tilde{v}_* := \sum_{i \in \mathbf{P}^*} \boldsymbol{\zeta}_{i,2}^{\lceil \log_2(t) \rceil}$$

Consistency Check:

6. Send $(\texttt{committed}, \mathsf{id}_h^{\text{com-zk}}, h)$ to every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{ZK}}^{n,R_{\mathsf{DL}}}$, using a fresh value for $\mathsf{id}_h^{\text{com-zk}}$. Wait until $(\texttt{com-proof}, \mathsf{id}_i^{\text{com-zk}}, r_i, R_i, \mathbf{P})$ is received (on behalf of $\mathcal{F}_{\mathsf{ZK}}^{n,R_{\mathsf{DL}}}$) from each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$. Let

$$R_* := \sum_{i \in \mathbf{P}^*} R_i$$
$$E^R := R_* - u_* \cdot G$$

7. If $e^k \neq 0$ or $E^R \neq 0$ or $k_* = 0$, then sample $k_h \leftarrow \mathbb{Z}_q$ and compute

   $$u_h := k_h \cdot k_* - u_* \qquad \text{and} \qquad R_h := u_h \cdot G \qquad \text{and} \qquad R := R_h + R_*$$

   Otherwise, send $(\texttt{inv}, \mathsf{id}^{\text{inv}}, \{h\} \cup \mathbf{P}^*, u_*)$ to $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ using a fresh value of id^{inv} on behalf of the corrupted parties. In return, receive $(\texttt{output}, \mathsf{id}^{\text{inv}}, R)$ and compute $R_h := R - R_*$. In either case, send $(\texttt{accept}, \mathsf{id}_h^{\text{com-zk}}, R_h)$ to every party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{ZK}}^{n,R_{\mathsf{DL}}}$.

8. On behalf of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$, wait to receive from $\mathcal{P}_i$ for every $i \in \mathbf{P}^*$ the decommitment message $(\texttt{decom-proof}, \mathsf{id}_i^{\text{com-zk}})$ corresponding to the commitments in Step 6. Upon receiving this message, abort if a valid witness was not input in Step 6. When all proofs are decommitted, send $(\texttt{exp-release}, \mathsf{id}^{\text{inv}})$ to $\mathcal{F}_{\mathsf{Inv}}^{n,t}$.

9. Send $(\mathtt{committed}, \mathsf{id}^{\mathsf{com}}_{h,2}, h)$ to every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^n_{\mathsf{Com}}$, using a fresh value for $\mathsf{id}^{\mathsf{com}}_{h,2}$. Wait to receive $(\mathtt{commit}, \mathsf{id}^{\mathsf{com}}_{i,2}, \Gamma^1_i)$ from every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}^n_{\mathsf{Com}}$. Compute

$$\Gamma^1_* \coloneqq \sum_{i \in \mathbf{P}^*} \Gamma^1_i$$

10. If $k_* = 0$, then abort by taking the following steps, and continue to Step 11 only if $k_* \neq 0$.

    (a) Instruct $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ to abort.

    (b) Sample $\phi_h \leftarrow \mathbb{Z}_q$, compute

    $$\Gamma^1_h \coloneqq \left(\left(\boldsymbol{\psi}_{*,2} \cdot \frac{\phi_h}{k_h} \right) - \tilde{v}_* \right) \cdot R$$

    and decommit these values to all parties indexed by $\mathbf{P}^*$ by sending $(\mathtt{decommitted}, \mathsf{id}^{\mathsf{com}}_{h,1}, \phi_h)$ and $(\mathtt{decommitted}, \mathsf{id}^{\mathsf{com}}_{h,2}, \Gamma^1_h)$ on behalf of $\mathcal{F}^n_{\mathsf{Com}}$. Note that per Step 7, k_h is always known if $k_* = 0$, and thus Γ^1_h is computable.

    (c) Wait for $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{h,1})$ and $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{h,2})$ from all parties indexed by $\mathbf{P}^*$, and then abort.

11. If $e^k \neq 0$ or $E^R \neq 0$ or $\tilde{v}_* \cdot R \neq \Gamma^1_*$ or $\phi_* = 0$, then some party has cheated. Abort by taking the following steps, and continue to Step 12 only if none of the above conditions hold.

    (a) Instruct $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ to abort.

    (b) Sample $\phi_h \leftarrow \mathbb{Z}_q$ and then compute

    $$E^{\Gamma^1} \coloneqq \frac{1}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k \right) \cdot E^R + k_* \cdot e^k \cdot G$$
    $$\Gamma^1_h \coloneqq \phi_h \cdot \left(\phi_* \cdot G + E^{\Gamma^1} \right) - \tilde{v}_* \cdot R$$

    and decommit these values to all parties indexed by $\mathbf{P}^*$ by sending $(\mathtt{decommitted}, \mathsf{id}^{\mathsf{com}}_{h,1}, \phi_h)$ and $(\mathtt{decommitted}, \mathsf{id}^{\mathsf{com}}_{h,2}, \Gamma^1_h)$ on behalf of $\mathcal{F}^n_{\mathsf{Com}}$. Note that in the equation that determines E^{Γ^1}, k_h has a coefficient of zero if $\phi_* = 0$, and that per Step 7, k_h is always known if $E^R \neq 0$ or $e^k \neq 0$. As we will show in the proof that follows this simulator, $\tilde{v}_* \cdot R \neq \Gamma^1_*$ implies at least one of the other conditions, and as a consequence, E^{Γ^1} is always computable.

    (c) Wait for $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{h,1})$ and $(\mathtt{decommit}, \mathsf{id}^{\mathsf{com}}_{h,2})$ from all parties indexed by $\mathbf{P}^*$, and then abort.

12. If $\tilde{v}_* = 0$, then send $(\texttt{inv-zero}, \mathsf{id}^{\mathsf{inv}})$ to $\mathcal{F}_{\mathsf{Inv}}^{n,t}$, receive $(\texttt{inv-output}, \mathsf{id}^{\mathsf{inv}}, 0)$ in response, and sample $\phi_h \leftarrow \mathbb{Z}_q$. Otherwise, send $(\texttt{inv-release}, \mathsf{id}^{\mathsf{inv}})$ to $\mathcal{F}_{\mathsf{Inv}}^{n,t}$, receive $(\texttt{inv-output}, \mathsf{id}^{\mathsf{inv}}, v_*)$, and compute

$$\phi_h := \frac{\tilde{v}_*}{v_* \cdot \phi_*}$$

Then, regardless of how ϕ_h is derived, compute

$$\Gamma_h^1 := \phi_h \cdot \phi_* \cdot G - \Gamma_*^1$$

and decommit ϕ_h and Γ_h^1 to all parties indexed by $\mathbf{P}^*$ by sending $(\texttt{decommitted}, \mathsf{id}_{h,1}^{\mathsf{com}}, \phi_h)$ and $(\texttt{decommitted}, \mathsf{id}_{h,2}^{\mathsf{com}}, \Gamma_h^1)$ on behalf of $\mathcal{F}_{\mathsf{Com}}^n$.

13. Wait for $(\texttt{decommit}, \mathsf{id}_{h,1}^{\mathsf{com}})$ and $(\texttt{decommit}, \mathsf{id}_{h,2}^{\mathsf{com}})$ from all parties indexed by $\mathbf{P}^*$, and then instruct $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ to release the appropriate output to $\mathcal{P}_h$, and terminate successfully.

Our proof proceeds as a sequence of hybrid experiments, starting from

$$\mathcal{H}_0 = \left\{ \mathsf{REAL}_{\pi_{\mathsf{Inv}}^{n,t}, \mathcal{A}^{\mathbf{P}^*}, \mathcal{Z}}(z) \right\}_{\substack{z \in \{0,1\}^*, \\ \mathbf{P}^* \subset \mathbf{P} : |\mathbf{P}^*| = t-1}}$$

The result of this experiment is characterized by the outputs u_i, v_i, and R for all parties indexed by $i \in [1, n]$, as well as the joint view of the parties indexed by $\mathbf{P}^*$, which is in turn characterized by the variables R_h, Γ_h^1, ϕ_h, $\boldsymbol{\zeta}_{i,1}^{\rho,h}$, and $\boldsymbol{\zeta}_{i,2}^{\rho,h}$.

Hybrid $\mathcal{H}_1$. This hybrid implements Steps 1 to 6 and 9 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, replacing Bob's instructions in Steps 2 to 6 and 8 of $\pi_{\mathsf{Inv}}^{n,t}$, and is otherwise identical to a real-world execution of $\pi_{\mathsf{Inv}}^{n,t}$. This is only a syntactic change, necessary to define variables that will be used by future hybrids, and thus the distribution of $\mathcal{H}_1$ is identical to that of $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This hybrid is identical to $\mathcal{H}_1$, except that we implement Step 10 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, and in addition, we implement the part of Step 7 that deals with the case that $k_* = 0$. $\mathcal{H}_2$ is distinguishable from $\mathcal{H}_1$ only by the fact that $\mathcal{H}_2$ aborts unconditionally when $k_* = 0$, whereas $\mathcal{H}_1$ does not, and by the distributions of ϕ_h and Γ_h^1, the only values released to the corrupt parties in the case that this abort is triggered.

First we consider the distributions of ϕ_h and Γ_h^1 when $k_* = 0$ and an abort is triggered. In both $\mathcal{H}_2$ and $\mathcal{H}_1$, ϕ_h is sampled uniformly, and the two are thus identically distributed in terms of this variable. In $\mathcal{H}_2$, per Step 10 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$,

$$\Gamma_h^1 = \left(\left(\frac{\phi_h}{k_h} \cdot \boldsymbol{\psi}_{*,2} \right) - \tilde{v}_* \right) \cdot R \tag{19}$$

and per Steps 3 through 5 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ and the specification of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$,

$$\tilde{v}_h + \tilde{v}_* = \frac{\phi_h}{k_h} \cdot \boldsymbol{\psi}_{*,2} \tag{20}$$

and thus by substitution of Equation 20 into Equation 19, we have $\Gamma_h^1 = \tilde{v}_h \cdot R$, which is exactly the distribution of Γ_h^1 in $\mathcal{H}_1$, per Step 8 of $\pi_{\mathsf{Inv}}^{n,t}$.

We now reason about the distributions of aborts when $k_* = 0$. $\mathcal{H}_2$ aborts unconditionally, and so an adversary could distinguish it from $\mathcal{H}_1$ by setting $k_* = 0$ and avoiding an abort. In $\mathcal{H}_1$, we have from Steps 3 to 5 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ and the specification of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$ that

$$u_h + u_* = k_h \cdot k_* \tag{21}$$

which implies that $u_h + u_* = 0$ regardless of the value of k_h. As a consequence $R = E^R$ per Step 6 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$. In order to avoid an abort, the adversary must arrange for the corrupted parties to send shares of Γ^1 such that

$$\begin{aligned} \Gamma^{1*} &= \phi_h \cdot \phi_* \cdot G - \Gamma_h^1 \\ &= \phi_h \cdot \phi_* \cdot G - \tilde{v}_h \cdot R \\ &= \phi_h \cdot \phi_* \cdot G - \tilde{v}_h \cdot E^R \end{aligned} \tag{22}$$

and by substituting Equation 20 into Equation 22 we have

$$\begin{aligned} \Gamma_*^1 &= \phi_h \cdot \phi_* \cdot G - \left(\frac{\phi_h}{k_h} \cdot \boldsymbol{\psi}_{*,2} - \tilde{v}_* \right) \cdot E^R \\ &= \phi_h \cdot \left(\phi_* \cdot G - \frac{\boldsymbol{\psi}_{*,2}}{k_h} \cdot E^R \right) + \tilde{v}_* \cdot E^R \end{aligned} \tag{23}$$

Recall that in $\mathcal{H}_1$, ϕ_h is sampled uniformly and independently of all other variables, and that it is hidden from the adversary until after the adversary has committed to Γ_*^1. The first term of Equation 23 is therefore uniform unless

$$\phi_* \cdot G - \frac{\boldsymbol{\psi}_{*,2}}{k_h} \cdot E^R = 0 \tag{24}$$

Now recall that both $\mathcal{H}_2$ and $\mathcal{H}_1$ abort with certainty when $\phi_* = 0$, and thus if the adversary is to distinguish the two hybrids, $1/k_h$ must have a nonzero coefficient. As with ϕ_h, the value k_h is sampled uniformly and independently of all other variables in $\mathcal{H}_1$, and is unknown to the adversary. When ϕ_*, E^R, and $\boldsymbol{\psi}_{*,2}$ are fixed, there is one value of k_h that satisfies Equation 24; thus over the coins of k_h we have $\Gamma_*^1 = \tilde{v}_* \cdot E^R$ with probability $1/q$ and otherwise Γ_*^1 is distributed uniformly in $\mathbb{G}$. Combining these two cases, we have $\Gamma_*^1 = \tilde{v}_* \cdot E^R$ with probability $2/q - 1/q^2$, and all other values occur with probability $1/q - 1/q^2$. Consequently an adversary can guess the correct value of Γ_*^1 and avoid an abort in $\mathcal{H}_1$ with probability at most $2/q - 1/q^2$, which is negligible in κ. $\mathcal{H}_1$ is thus statistically indistinguishable from $\mathcal{H}_2$.

Hybrid $\mathcal{H}_3$. This hybrid is identical to $\mathcal{H}_2$, except that we remove Step 10 of $\pi_{\mathsf{Inv}}^{n,t}$ and instead implement Step 11 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ and the part of Step 7 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ that deals with the case that $e^k \neq 0$ or $E^R \neq 0$. Consequently, this hybrid differs

from $\mathcal{H}_2$ according to the distributions of ϕ_h and Γ_h^1 (the only values released to the corrupt parties in the case of an abort) and according to the distributions of aborts: $\mathcal{H}_3$ aborts unconditionally when $e^k \neq 0$ or $E^R \neq 0$ or $\tilde{v}_* \cdot R \neq \Gamma_*^1$, whereas $\mathcal{H}_2$ aborts unconditionally when $\Gamma_h^1 + \Gamma_*^1 \neq \phi_h \cdot \phi_* \cdot G$.

We first assume an abort has been triggered and consider the distributions of ϕ_h and Γ_h^1. In both $\mathcal{H}_3$ and $\mathcal{H}_2$, ϕ_h is sampled uniformly, and the two are thus identically distributed in terms of this variable. In $\mathcal{H}_3$ per Step 11 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$,

$$E^{\Gamma^1} = \frac{1}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k\right) \cdot E^R + k_* \cdot e^k \cdot G \tag{25}$$

$$\Gamma_h^1 = \phi_h \cdot \left(\phi_* \cdot G + E^{\Gamma^1}\right) - \tilde{v}_* \cdot R \tag{26}$$

and by expanding Equation 25, we have

$$E^{\Gamma^1} = \frac{\phi_*}{k_h \cdot k_*} \cdot E^R + \frac{e^k}{k_h} \cdot E^R + \frac{k_h \cdot k_* \cdot e^k}{k_h} \cdot G$$

to both sides of which we can add $\phi_* \cdot G$, yielding

$$\begin{aligned}
\phi_* \cdot G + E^{\Gamma^1} &= \frac{\phi_*}{k_h \cdot k_*} \cdot E^R + \frac{\phi_* \cdot k_h \cdot k_*}{k_h \cdot k_*} \cdot G + \frac{e^k}{k_h} \cdot E^R + \frac{k_h \cdot k_* \cdot e^k}{k_h} \cdot G \\
&= \frac{\phi_*}{k_h \cdot k_*} \cdot \left(k_h \cdot k_* \cdot G + E^R\right) + \frac{e^k}{k_h} \cdot \left(k_h \cdot k_* \cdot G + E^R\right) \\
&= \left(\frac{\phi_*}{k_h \cdot k_*} + \frac{e^k}{k_h}\right) \cdot \left(k_h \cdot k_* \cdot G + E^R\right)
\end{aligned}$$

which, via substitution of Equation 21 yields

$$\phi_* \cdot G + E^{\Gamma^1} = \left(\frac{\phi_*}{k_h \cdot k_*} + \frac{e^k}{k_h}\right) \cdot \left((u_h + u_*) \cdot G + E^R\right) \tag{27}$$

We also have from Steps 6 and 7 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ that

$$(u_h + u_*) \cdot G + E^R = R \tag{28}$$

and per Steps 3 through 5 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ and the specification of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$,

$$\frac{1}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k\right) = \frac{\tilde{v}_h + \tilde{v}_*}{\phi_h} \tag{29}$$

and by substitution of Equations 28 and 29 into 27 we have

$$\phi_* \cdot G + E^{\Gamma^1} = \frac{1}{k_h} \left(\frac{\phi_*}{k_*} + e^k\right) \cdot R = \frac{\tilde{v}_h + \tilde{v}_*}{\phi_h} \cdot R \tag{30}$$

Finally, by substituting Equation 30 into Equation 26 we find that $\Gamma_h^1 = \tilde{v}_h \cdot R$, which is exactly the distribution of Γ_h^1 in $\mathcal{H}_2$, per Step 8 of $\pi_{\mathsf{Inv}}^{n,t}$. Thus the

distributions of ϕ_h and Γ_h^1 are identical in $\mathcal{H}_3$ and $\mathcal{H}_2$, and it remains only to reason about the distribution of aborts.

In both $\mathcal{H}_3$ and $\mathcal{H}_2$, an abort is guaranteed if $\phi_* = 0$. In $\mathcal{H}_3$, an abort is also guaranteed if $e^k \neq 0$ or $E^R \neq 0$ or $\tilde{v}_* \cdot R \neq \Gamma_*^1$. On the other hand, in $\mathcal{H}_2$, an abort occurs if $\Gamma_h^1 + \Gamma_*^1 \neq \phi_h \cdot \phi_* \cdot G$. Thus, an adversary can distinguish $\mathcal{H}_3$ from $\mathcal{H}_2$ if they can set $e^k \neq 0$ or $E^R \neq 0$ or $\tilde{v}_* \cdot R \neq \Gamma_*^1$ and still transmit values of ϕ_* and Γ_*^1 such that $\Gamma_h^1 + \Gamma_*^1 = \phi_h \cdot \phi_* \cdot G$, or if they can set $e^k = 0$ and $E^R = 0$ and $\tilde{v}_* \cdot R = \Gamma_*^1$ and transmit values of ϕ_* and Γ_*^1 such that $\Gamma_h^1 + \Gamma_*^1 \neq \phi_h \cdot \phi_* \cdot G$. We will show that the latter circumstance is impossible, and that the former occurs with negligible probability.

As previously given in Equations 25 and 26, we have in both $\mathcal{H}_2$ and $\mathcal{H}_3$ that

$$\Gamma_h^1 = \phi_h \cdot \left(\phi_* + k_* \cdot e^k\right) \cdot G + \frac{\phi_h}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k\right) \cdot E^R - \tilde{v}_* \cdot R \tag{31}$$

Notice that if $e^k = 0$ and $E^R = 0$ and $\tilde{v}_* \cdot R = \Gamma_*^1$, then it follows directly from Equation 31 that $\Gamma_h^1 + \Gamma_*^1 = \phi_h \cdot \phi_* \cdot G$, and that $\mathcal{H}_2$ cannot abort unless $\mathcal{H}_3$ also aborts. It remains to consider whether an adversary can avoid an abort in $\mathcal{H}_2$ while setting $e^k \neq 0$ or $E^R \neq 0$ or $\tilde{v}_* \cdot R \neq \Gamma_*^1$. According to Step 10 of $\pi_{\mathsf{Inv}}^{n,t}$ and Equation 31, an abort can by avoided in $\mathcal{H}_2$ only if the corrupted parties transmit shares of the check value such that

$$\Gamma_*^1 = \tilde{v}_* \cdot R - \phi_h \cdot \left(\frac{1}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k\right) \cdot E^R - k_* \cdot e^k \cdot G\right) \tag{32}$$

Notice that Equation 32 implies that $\tilde{v}_* \cdot R \neq \Gamma_*^1$ only if $e^k \neq 0$ or $E^R \neq 0$; it is therefore sufficient to consider whether an adversary can set $e^k \neq 0$ or $E^R \neq 0$ and yet avoid an abort in $\mathcal{H}_2$. Recall that ϕ_h is sampled uniformly and independently of all other variables, and that it is hidden from the corrupt parties until after they are committed to Γ_*^1. Thus, $\Gamma_*^1 = \tilde{v}_* \cdot R$ if

$$\frac{1}{k_h} \cdot \left(\frac{\phi_*}{k_*} + e^k\right) \cdot E^R + k_* \cdot e^k \cdot G = 0 \tag{33}$$

and otherwise Γ_*^1 is distributed uniformly in $\mathbb{G}$. Recall that k_* must be nonzero because both $\mathcal{H}_3$ and $\mathcal{H}_2$ abort with certainty when $k_* = 0$. Combining this fact with the fact that $e^k \neq 0$ or $E^R \neq 0$, we find that $1/k_h$ must have a nonzero coefficient in Equation 33 if the two hybrids are to be distinguished. Observe that E^R, ϕ_*, k_*, and e^k are fixed before k_h is chosen. For each assignment of E^R, ϕ_*, k_*, and e^k, there is exactly one value of k_h that satisfies Equation 33, and consequently, over the (uniform) coins of k_h we have $\Gamma_*^1 = \tilde{v}_* \cdot R$ with probability $1/q$ and otherwise Γ_*^1 is distributed uniformly in $\mathbb{G}$. Combining these two cases, we have $\Gamma_*^1 = \tilde{v}_* \cdot R$ with probability $2/q - 1/q^2$, and all other values occur with probability $1/q - 1/q^2$. Consequently an adversary can guess the correct value of Γ_*^1 and avoid an abort in $\mathcal{H}_2$ with probability at most $2/q - 1/q^2$, which is negligible in κ. $\mathcal{H}_2$ is thus statistically indistinguishable from $\mathcal{H}_3$.

Hybrid $\mathcal{H}_4$. $\mathcal{H}_4$ differs from $\mathcal{H}_3$ in that it implements Steps 7, 8, 12, and 13 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, replacing $\mathbf{P}_h$'s instructions in Steps 1, 7, 9, and 11 of $\pi_{\mathsf{Inv}}^{n,t}$. Thus,

$$\mathcal{H}_4 = \left\{ \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{Inv}}^{n,t},(\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*},\mathcal{A}^{\mathbf{P}^*}),\mathcal{Z}}(z) \right\}_{\substack{z\in\{0,1\}^*,\\ \mathbf{P}^*\subset\mathbf{P}:|\mathbf{P}^*|=t-1}}$$

The view of the adversary is characterized by the values R_h, ϕ_h, and Γ_h^1, and so we must consider the distributions of these variables jointly with the output values u_h and v_h in order to argue indistinguishability.

In $\mathcal{H}_3$, when $e^k = 0$ and $E^R = 0$, R_h is calculated per Step 6 of $\pi_{\mathsf{Inv}}^{n,t}$ as $R_h = u_h \cdot G$. Because the honest party samples its instance key share k_h uniformly in Step 1 of $\pi_{\mathsf{Inv}}^{n,t}$, Equation 21 implies that u_h is uniform and independent of the other variables in the adversary's view, and the uniformity of R_h follows. In $\mathcal{H}_4$, $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ chooses u_h uniformly, and returns $R := R_* + u_h \cdot G$, from which the simulator calculates R_h by subtraction. Thus, the distributions of u_h and R_h are identical in $\mathcal{H}_4$ and $\mathcal{H}_3$.

In $\mathcal{H}_3$, ϕ_h is sampled uniformly, and given that value the adversary computes

$$v_* = \frac{\tilde{v}_*}{\phi_h \cdot \phi_*} \tag{34}$$

as specified in Step 11 of $\pi_{\mathsf{Inv}}^{n,t}$. From Equations 34, 21, and 20, and the fact that $e^k = 0$ and $\phi_* \neq 0$, we find that in $\mathcal{H}_3$

$$v_h = \frac{1}{u_h + u_*} - v_* \tag{35}$$

In $\mathcal{H}_4$, on the other hand, Step 12 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ implies that v_h and ϕ_h may be set in two different ways, depending on the adversary's choice of $\tilde{v}_*$. In the case that $\tilde{v}_* = 0$, ϕ_h is sampled uniformly by $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$, and $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ is signaled via the `inv-zero` interface, which causes it to set

$$v_h = \frac{1}{u_h + u_*}$$

Thus, when $\tilde{v}_* = 0$, both ϕ_h and v_h have identical distributions in $\mathcal{H}_4$ and $\mathcal{H}_3$.

In the case that $\tilde{v}_* \neq 0$, $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ uses the `inv-output` interface to retrieve a value v_* from $\mathcal{F}_{\mathsf{Inv}}^{n,t}$, which is uniform, but adheres to the relation

$$v_h + v_* = \frac{1}{u_h + u_*}$$

which is equivalent to Equation 35. ϕ_h is calculated per Step 12 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$ as

$$\phi_h = \frac{\tilde{v}_*}{v_* \cdot \phi_*}$$

which is equivalent to Equation 34. Thus, regardless of the value of $\tilde{v}_*$, both ϕ_h and v_h have distributions identical to their distributions in $\mathcal{H}_3$.

The final value to consider is Γ_h^1. In $\mathcal{H}_3$, it is computed per Step 8 of $\pi_{\mathsf{Inv}}^{n,t}$ as

$$\Gamma_h^1 = \tilde{v}_h \cdot R \tag{36}$$

and because $e^k = 0$, we have from Equations 21 and 20 that

$$\tilde{v}_h + \tilde{v}_* = \frac{\phi_h \cdot \phi_*}{u_h + u_*}$$

which we substitute into Equation 36 to find that

$$\Gamma_h^1 = \left(\frac{\phi_h \cdot \phi_*}{u_h + u_*} - \tilde{v}_* \right) \cdot R \tag{37}$$

Next, recall that per Steps 6 and 7 of $\mathcal{S}_{\mathsf{Inv}}^{n,t,\mathbf{P}^*}$,

$$R = (u_h + u_*) \cdot G + E^R \tag{38}$$

and recall that $E^R = 0$ and $\Gamma_*^1 = \tilde{v}_* \cdot R$. Substituting Equation 38 into 37 yields

$$\Gamma_h^1 = \phi_h \cdot \phi_* \cdot G - \tilde{v}_* \cdot R = \phi_h \cdot \phi_* \cdot G - \Gamma_*^1$$

which is exactly how Γ_h^1 is calculated in $\mathcal{H}_4$. Thus the distributions of all variables are identical in $\mathcal{H}_4$ and $\mathcal{H}_3$, and the two hybrids are perfectly indistinguishable. By transitivity, we have also that $\mathcal{H}_4 \overset{s}{\equiv} \mathcal{H}_0$, and thus Theorem 4.1 is proved. □

D Proof of Security for t-of-n ECDSA

In this section, we reduce the security of our setup and signing protocols to the difficulty of solving the Computational Diffie-Hellman problem in $\mathbb{G}$. As we are concerned with the security of our threshold signing system over the lifetime of a public key, and in consideration of the interactions of all participating parties, we specify a shell protocol $\pi_{\mathsf{ECDSA}}^{n,t}$ which orchestrates a signing *epoch*, in which the parties perform a single setup as a group, followed by some number of signatures between subgroups of size t. It is with respect to this shell protocol that we claim security, and because we claim only static security, we predetermine both the set of messages to be signed and the subgroups of parties who will participating in the signing process for each.

Protocol 5. *t*-of-*n* ECDSA $\left(\pi_{\mathsf{ECDSA}}^{n,t}\right)$:

This protocol is parameterized by the union of the parameters of its subprotocols; specifically, by the party count n, the threshold size t, the elliptic curve group $(\mathbb{G}, G, q)$, and the statistical security parameter s. It receives as input a vector $\mathbf{m} \in \{0,1\}^{*\times *}$ of messages and a vector $\mathfrak{P}$ of groups of parties to sign those messages, such that each group $\mathfrak{P}_j \subset [1,n]$ is of size t. To each party, it outputs a vector of signatures.

Setup:

1. The parties jointly run $\pi_{\mathsf{ECDSA\text{-}Setup}}^{n,t}$ with no inputs. Each party $\mathcal{P}_i$ receives as output the joint public key pk and a point $p(i)$ on the polynomial p.

Signing:

2. For each message $\mathbf{m}_j$ in $\mathbf{m}$, let $\mathfrak{P}_j$ be the group of t parties associated with that message. Now this group runs $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ with $\mathbf{P} = \mathfrak{P}_j$, each party $\mathcal{P}_i$ for $i \in \mathbf{P}$ supplying $p(i)$ and all parties supplying $\mathbf{m}_j$ and receiving the signature as output.

Theorem 5.1. *Let $(\mathbb{G}, \mathbf{G}, \mathbf{q})$ be an infinite sequence of elliptic curves in which the Computational Diffie-Hellman Problem is hard. The protocol $\pi^{n,t}_{\mathsf{ECDSA}}$ UC-realizes the functionality $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ for this sequence in the $(\mathcal{F}^{n,t}_{\mathsf{Inv}}, \mathcal{F}^{\ell}_{\mathsf{2PMul}}, \mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}, \mathcal{F}^{n}_{\mathsf{Com}})$-hybrid model, in the presence of a malicious adversary statically corrupting up to $t-1$ parties.*

Proof. We make the simplifying assumption that the adversary makes complete use of its power by corrupting exactly $t-1$ parties with indices given by $\mathbf{P}^*$. Our proof will be via a sequence of hybrid experiments, beginning with

$$\mathcal{H}_0 = \left\{ \mathsf{REAL}_{\pi^{n,t}_{\mathsf{ECDSA}}, \mathcal{A}^{\mathbf{P}^*}, \mathcal{Z}}(z) \right\}_{\substack{z \in \{0,1\}^*, \\ \mathbf{P}^* \subset \mathbf{P} : |\mathbf{P}^*| = t-1}}$$

Due to the length and complexity of this proof, we have divided it into two sections. In Appendix D.1, we give a hybrid in which the components of the corrupt parties' transcripts that are due to the setup protocol $\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$ are simulated. In Appendix D.2, we give a further sequence of hybrids that replace the transcript components due to participation in the signing protocol $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. We begin by giving a master simulator, which corresponds to $\pi^{n,t}_{\mathsf{ECDSA}}$ and calls upon the simulators we introduce in subsequent sections.

Simulator 4. t-of-n ECDSA against $\mathbf{P}^*$ $\left(\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA}}\right)$:

This simulator interposes between a group of $t-1$ malicious parties $\mathbf{P}^* \subset [1,n]$ and the ideal functionality $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$. It is parameterized by the union of the parameters of the simulators that it calls; specifically, by the party count n, the threshold t, the elliptic curve $(\mathbb{G}, G, q)$, and the statistical security parameter s. It receives as input a vector of messages $\mathbf{m} \in \{0,1\}^{*\times *}$ and a vector $\mathfrak{P}$ of party groups with which those messages should be signed.

Setup:

1. Run $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Setup}}$ against the parties indexed by $\mathbf{P}^*$ and receive the public key pk and a secret key share $p(i)$ for $i \in \mathbf{P}^*$.

Signing:

2. For each item $\mathbf{m}_j$ in $\mathbf{m}$, invoke $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ against $\mathbf{P}^* \cap \mathfrak{P}_j$ with message $\mathbf{m}_j$, public key pk, a fresh signature index $\mathsf{id}^{\mathsf{sig}}$, and the appropriate secret key shares $p(i)$ for $i \in \mathbf{P}^* \cap \mathfrak{P}_j$.

D.1 Simulating Setup

Simulator 5. t-of-n ECDSA Setup against $\mathbf{P}^*$ ($\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Setup}}$):

This simulator interposes between an evaluation of the $\pi^{n,t}_{\mathsf{ECDSA\text{-}Setup}}$ protocol involving the group of $t-1$ malicious parties indexed by $\mathbf{P}^*$ and the ideal functionality $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$. It acts on behalf of of the remaining, non-corrupted parties, who interact with the functionality. For convenience, let us index these parties by $\overline{\mathbf{P}}^* = [1,n] \setminus \mathbf{P}^*$. $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Setup}}$ receives no inputs, and it extracts and returns $\mathcal{P}_i$'s share of the secret key $p(i)$ for all $i \in \mathbf{P}^*$. It is parameterized by the elliptic curve $(\mathbb{G}, G, q)$, and it plays the role of $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ and $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$ in their interactions with the parties indexed by $\mathbf{P}^*$, and the role of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ in its interaction with the adversary.

Public Key Generation:

1. For all $i \in \mathbf{P}^*$ and $j \in \overline{\mathbf{P}}^*$, sample $p_j(i) \leftarrow \mathbb{Z}_q$ and send $p_j(i)$ to party $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$. Receive $p_{i,j}(j)$ from $\mathcal{P}_i$ in response.

2. For all $i \in \mathbf{P}^*$ and $j \in \overline{\mathbf{P}}^*$, send $(\texttt{committed}, j)$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$.

3. For all $i \in \mathbf{P}^*$, receive $(\texttt{com-proof}, \mathsf{id}_i^{\mathsf{com\text{-}zk}}, p(i), T_i)$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}$.

4. Let p_* be the joint polynomial chosen (and distributed) by the corrupt parties; that is, the sum of their individual polynomials. We wish to check whether this polynomial indeed has degree $t-1$ as specified by the protocol. For $i \in \mathbf{P}^*$, calculate

$$p_*(i) \coloneqq p(i) - \sum_{j \in \overline{\mathbf{P}}^*} p_j(i)$$

   where $p(i)$ is the value that $\mathcal{P}_i$ committed in Step 3 and $p_j(i)$ is the value sent to $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ for $j \in \overline{\mathbf{P}}^*$ in Step 1. For each honest party $j \in \overline{\mathbf{P}}^*$ calculate

$$p_*(j) \coloneqq \sum_{i \in \mathbf{P}^*} p_{i,j}(j)$$

   where $p_{i,j}(j)$ is the value received from $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ in Step 1. The values $p_*(i)$ $\forall i \in [1,n]$ define a *consistent* polynomial of degree $t-1$ if and only if for all $x \in [1, n-t-1]$ defining $\mathbf{J}^x = [x, x+t]$ and $\mathbf{J}^{x+1} = [x+1, x+t+1]$,

$$\sum_{j \in \mathbf{J}^x} \lambda_j^{\mathbf{J}^x}(0) \cdot T_j = \sum_{j \in \mathbf{J}^{x+1}} \lambda_j^{\mathbf{J}^{x+1}}(0) \cdot T_j$$

where $\lambda_j^{\mathbf{J}^x}(y)$ and $\lambda_j^{\mathbf{J}^{x+1}}(y)$ are party $\mathcal{P}_j$'s Lagrange coefficients for interpolating p at location y with the sets of parties indexed by $\mathbf{J}^x$ and $\mathbf{J}^{x+1}$ respectively.

5. If $p_*(i)$ is *not* a consistent polynomial of degree $t-1$ or there is some $i \in \mathbf{P}^*$ for which $T_i \neq p(i) \cdot G$, then use the following steps to run the protocol as honest parties would, and afterward abort. Proceed to Step 6 only if $p_*(i)$ is a consistent polynomial of degree $t-1$.

   (a) For $j \in \overline{\mathbf{P}}^*$, sample $p_j(0) \leftarrow \mathbb{Z}_q$, and then for $j, h \in \overline{\mathbf{P}}^*$, calculate party $\mathcal{P}_h$'s point on $\mathcal{P}_j$'s polynomial

   $$p_j(h) := \frac{p_j(0)}{\lambda_h^{\{h\}\cup\mathbf{P}^*}(0)} - \sum_{i\in\mathbf{P}^*} \frac{\lambda_i^{\{h\}\cup\mathbf{P}^*}(0)}{\lambda_h^{\{h\}\cup\mathbf{P}^*}(0)} \cdot p_j(i)$$

   (b) For $j \in \overline{\mathbf{P}}^*$, calculate party $\mathcal{P}_j$'s point on the joint polynomial

   $$p(j) := \sum_{i\in\overline{\mathbf{P}}^*} p_i(j) + \sum_{i\in\mathbf{P}^*} p_{i,j}(j)$$

   (c) For $j \in \overline{\mathbf{P}}^*$, calculate

   $$T_j := p(j) \cdot G$$

   and send $(\texttt{accept}, j, T_j)$ to $\mathcal{P}_i$ for all $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$.

   (d) For all $i \in \mathbf{P}^*$, receive $(\texttt{decom-proof})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$.

   (e) Abort.

6. Send $(\texttt{init})$ to the $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$ functionality and receive $(\texttt{public-key}, \mathsf{pk})$ in response.

7. For $j \in \overline{\mathbf{P}}^*$, calculate

   $$T_j := \frac{\mathsf{pk}}{\lambda_j^{\{j\}\cup\mathbf{P}^*}(0)} - \sum_{i\in\mathbf{P}^*} \frac{\lambda_i^{\{j\}\cup\mathbf{P}^*}(0)}{\lambda_j^{\{j\}\cup\mathbf{P}^*}(0)} \cdot p(i) \cdot G$$

   and send $(\texttt{accept}, j, T_j)$ to $\mathcal{P}_i$ for all $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$.

8. For all $i \in \mathbf{P}^*$, receive $(\texttt{decom-proof})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}},n}$.

9. If $T_i = p(i) \cdot G$ for all $i \in \mathbf{P}^*$, then output $p(i)$ as $\mathcal{P}_i$'s extracted secret key share and, after Step 10, terminate successfully. Otherwise, abort.

Auxiliary Setup:

10. Receive message (`init`) from the adversary on behalf of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$, and for all $i \in \mathbf{P}^*$, receive message (`init`) from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$. Send (`init-complete`) to $\mathcal{P}_i$ as appropriate.

Hybrid $\mathcal{H}_1$. This hybrid experiment is the same as $\mathcal{H}_0$, except that $\mathcal{S}_{\mathsf{ECDSA\text{-}Setup}}^{n,t,\mathbf{P}^*}$ is completely implemented, replacing all components derived from the invocation of $\pi_{\mathsf{ECDSA\text{-}Setup}}^{n,t}$ in the transcripts of all corrupted parties. The polynomials of the honest parties are uniformly sampled in both $\mathcal{H}_0$ and $\mathcal{H}_1$. Following the reception of the corrupted parties' polynomial points, there are two cases $\mathcal{H}_1$, defined by the abort condition in Step 5 of $\mathcal{S}_{\mathsf{ECDSA\text{-}Setup}}^{n,t,\mathbf{P}^*}$. In the first case, there exists $x \in [1, n-t-1]$ such that

$$\sum_{j \in \mathbf{J}^x} \lambda_j^{\mathbf{J}^x}(0) \cdot T_j \neq \sum_{j \in \mathbf{J}^{x+1}} \lambda_j^{\mathbf{J}^{x+1}}(0) \cdot T_j$$

and the simulation aborts. It is trivially true that $\pi_{\mathsf{ECDSA\text{-}Setup}}^{n,t}$ also aborts in this case, since the honest parties perform exactly the same test in Step 6 of $\pi_{\mathsf{ECDSA\text{-}Setup}}^{n,t}$, and because the simulator follows the protocol's instructions exactly in this case, the messages it sends are identically distributed to their counterparts in $\mathcal{H}_0$. In the second case,

$$\sum_{j \in \mathbf{J}^x} \lambda_j^{\mathbf{J}^x}(0) \cdot T_j = \sum_{j \in \mathbf{J}^{x+1}} \lambda_j^{\mathbf{J}^{x+1}}(0) \cdot T_j \tag{39}$$

for all $x \in [1, n-t-1]$, no abort occurs in either $\mathcal{H}_1$ or $\mathcal{H}_0$, and the views of the corrupted parties are characterized by the values T_j for $j \in \overline{\mathbf{P}}^*$. Note that in both hybrids, Equation 39 implies that these values are completely constrained by T_i for $i \in \mathbf{P}^*$ and pk. In $\mathcal{H}_0$, we know that pk is uniform by virtue of containing an additive, uniform contribution from at least one honest party. In $\mathcal{H}_1$, $\mathcal{S}_{\mathsf{ECDSA\text{-}Setup}}^{n,t,\mathbf{P}^*}$ retrieves a uniform pk from $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$, and uses Lagrange interpolation to reconstruct the appropriate values of T_j. Thus $\mathcal{H}_1$ and $\mathcal{H}_0$ are identically distributed.

D.2 Simulating Signing

In this subsection, as in our proof of Theorem 4.1, we make the additional simplifying assumption that the $t-1$ parties corrupted by the adversary are chosen such that during the signing procedure, $\mathbf{P}^* \subset \mathbf{P}$, and only a single honest party with index h remains.

Before we give a formal description of our simulator and hybrids, we informally outline our argument, which is structurally similar to our argument for $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ and $\pi_{\mathsf{Inv}}^{n,t}$. The adversary's view is characterized by the values Γ_h^2, Γ_h^3, and sig_h, along with the pattern of aborts. Because the simulator plays the roles of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$ and $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, it can observe the adversary's inputs and outputs in the Multiplication and Inversion phase of $\pi_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}}$ and use this information to choose one of two

simulation strategies for the subsequent phases. If the adversary's behavior is consistent with an honest execution of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, then the simulator queries $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ for the nonce R, and because the adversary has behaved honestly, Γ^2_h and Γ^3_h can be calculated as linear functions of its outputs from $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ and $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. Likewise, the expected values of Γ^2_i and Γ^3_i for $i \in \mathbf{P}^*$ can be calculated, and the simulator aborts if unexpected ones are received. Finally, the simulator queries $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ for the signature, and sig_h is computed as a linear function of its response and values already known. Thus, when the adversary is honest in the Multiplication and Inversion phase of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, the simulation is perfect.

On the other hand, when the adversary behaves dishonestly in the Multiplication and Inversion phase of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, the simulator samples its own instance key without querying $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$, and then aborts unconditionally in the Consistency Check phase. The simulator effectively runs the code of an honest party, and so the messages it sends before it aborts are distributed identically to those of an honest party. The simulation is thus distinguishable from the real world only if the adversary can behave dishonestly and yet avoid an abort in the real world. We argue that achieving this is as hard as solving the Computational Diffie-Hellman Problem.

This simulator can be generalized to the case of acting on behalf of multiple honest parties via the same transformation we gave for $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{Inv}}$: for each additional honest party, the simulator runs an instance of the code in $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ that interacts with the corrupt parties, and for the purpose of simulating party h, these virtual parties are treated adversarially.

Simulator 6. *t*-of-*n* ECDSA Signing against $\mathbf{P}^*$ $\left(\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}\right)$:

This simulator interposes between an evaluation of the $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ protocol involving the group of $t-1$ malicious parties indexed by $\mathbf{P}^*$ and the ideal functionality $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$. It acts on behalf of a single honest party $\mathcal{P}_h$, who interacts with the functionality. $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ receives as input a message m, the signature id $\mathsf{id}^{\mathsf{sig}}$, the public key pk, and $\mathcal{P}_i$'s share of the secret key $p(i)$ for all $i \in \mathbf{P}^*$; it extracts and returns the signature output shared by all parties. It is parameterized by the elliptic curve $(\mathbb{G}, G, q)$ and the statistical security parameter s, and it plays the roles of $\mathcal{F}^{n}_{\mathsf{Com}}$ and $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$ in their interactions with the parties indexed by $\mathbf{P}^*$, and the role of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ in its interactions with the adversary.

Multiplication and Inversion:

1. On behalf of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$, receive $(\texttt{inv}, \mathsf{id}^{\mathsf{inv}}, \mathbf{P}, u_*)$ from the adversary, and define h such that $\mathbf{P} = \{h\} \cup \mathbf{P}^*$.

2. Send $(\texttt{get-instance-key}, \mathsf{id}^{\mathsf{sig}})$ to $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ on behalf of each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ and in response receive $(\texttt{instance-key}, \mathsf{id}^{\mathsf{sig}}, R)$. Send $(\texttt{exp-output}, \mathsf{id}^{\mathsf{inv}}, R)$ to the corrupted parties on behalf of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$.

3. Receive either $(\texttt{inv-release}, \mathsf{id}^{\mathsf{inv}})$ or $(\texttt{inv-zero}, \mathsf{id}^{\mathsf{inv}})$ from the adversary

on behalf of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$. If $(\texttt{inv-zero}, \mathsf{id}^{\mathsf{inv}})$ is received, set $v_* := 0$. Otherwise, sample $v_* \leftarrow \mathbb{Z}_q$. Regardless, send $(\texttt{inv-output}, \mathsf{id}^{\mathsf{inv}}, v_*)$ to the adversary on behalf of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$.

4. On behalf of $\mathcal{F}_{\mathsf{2PMul}}^{\ell}$, perform the appropriate preprocessing steps for two party multiplication between $\mathcal{P}_h$ and each party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$. Next, receive $(\texttt{input}, \mathsf{id}_{i,2}^{\mathsf{mul}}, \{\mathsf{sk}_i, v_i\})$ from each party $\mathcal{P}_i$ and respond with $(\texttt{output}, \mathsf{id}_{i,2}^{\mathsf{mul}}, \{w_i^{h,1}, w_i^{h,2}\})$ where $w_i^{h,1} \leftarrow \mathbb{Z}_q$ and $w_i^{h,2} \leftarrow \mathbb{Z}_q$.

5. Compute the true joint secret key and the joint output share of the corrupt parties

$$\mathsf{sk}_* := \sum_{i \in \mathbf{P}^*} \lambda_i^{\{h\} \cup \mathbf{P}^*}(0) \cdot p(i)$$
$$w_* := \mathsf{sk}_* \cdot v_* + \sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2} \right)$$

   and then use the former value to compute the error terms

$$e^{\mathsf{sk}} := \sum_{i \in \mathbf{P}^*} \mathsf{sk}_i - \mathsf{sk}_*$$
$$e^{v} := \sum_{i \in \mathbf{P}^*} v_i - v_*$$

   If either e^{sk} or e^v is nonzero, then abort in Step 7.

Consistency Check:

6. Send $(\texttt{committed}, \mathsf{id}_h^{\mathsf{com}})$ to every party $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}^{n}$ and wait for $(\texttt{commit}, \mathsf{id}_i^{\mathsf{com}}, \left(\Gamma_i^2, \Gamma_i^3\right), \mathbf{P})$ such that $h \in \mathbf{P}$ in response.

7. If either of the error terms e^{sk} or e^v is nonzero, then abort via the following procedure. Continue to Step 8 only if both of these terms are zero.

   (a) Sample u_h and v_h uniformly subject to

$$u_h + u_* = \frac{1}{v_h + v_*}$$

   (b) Compute the check values

$$\Gamma_h^2 := \left(w_* + \mathsf{sk}_* \cdot e^v - v_h \cdot e^{\mathsf{sk}} \right) \cdot G - (v_* + e^v) \cdot \mathsf{pk}$$

$$\Gamma_h^3 \coloneqq (1 + e^v \cdot (u_h + u_*)) \cdot \mathsf{pk} + \left(v_h \cdot e^{\mathsf{sk}} - \mathsf{sk}_* \cdot e^v - w_*\right) \cdot R$$

and send $(\texttt{decommitted}, \mathsf{id}_h^{\mathsf{com}}, (\Gamma_h^2, \Gamma_h^3))$ to every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}^n$.

(c) Abort by sending $(\texttt{abort}, \mathsf{id}^{\mathsf{sig}})$ to $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$.

8. Compute the check values

$$\Gamma_h^2 \coloneqq w_* \cdot G - v_* \cdot \mathsf{pk}$$
$$\Gamma_h^3 \coloneqq \mathsf{pk} - w_* \cdot R$$

and send $(\texttt{decommitted}, \mathsf{id}_h^{\mathsf{com}}, (\Gamma_h^2, \Gamma_h^3))$ to every $\mathcal{P}_i^*$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{Com}}^n$.

9. If

$$\sum_{j \in \{h\} \cup \mathbf{P}^*} \Gamma_j^2 \neq 0 \qquad \text{or} \qquad \sum_{j \in \{h\} \cup \mathbf{P}^*} \Gamma_j^3 \neq \mathsf{pk}$$

then abort by sending $(\texttt{abort}, \mathsf{id}^{\mathsf{sig}})$ to $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$.

Signing:

10. Send $(\texttt{proceed}, \mathsf{id}^{\mathsf{sig}}, i)$ to $\mathcal{F}_{\mathsf{ECDSA}}^{n,t}$ on behalf of each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ and receive $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \mathsf{sig})$ in response.

11. Compute $\mathcal{P}_h$'s share of the final signature

$$\mathsf{sig}_* \coloneqq H(m) \cdot v_* + r_x \cdot w_*$$
$$\mathsf{sig}_h \coloneqq \mathsf{sig} - \mathsf{sig}_*$$

and send sig_h to every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$ on behalf of $\mathcal{P}_h$, and halt successfully.

Hybrid $\mathcal{H}_2$. This hybrid implements Steps 1 through 6 of $\mathcal{S}_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}^*}$, as well as Step 7, but conditioned *only* on $e^{\mathsf{sk}} \neq 0$, replacing $\mathcal{P}_h$'s instructions in Steps 1 through 4 of $\pi_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}}$, and replacing Step 5 in the case that an abort is triggered. Apart from the partial implementation of Step 7 of $\mathcal{S}_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}^*}$, these changes are purely syntactical, and so this hybrid differs from $\mathcal{H}_1$ according to the fact that it aborts unconditionally when

$$\sum_{i \in \mathbf{P}^*} \mathsf{sk}_i \neq \mathsf{sk}_*$$

and according to the distributions of the values Γ_h^2 and Γ_h^3 when an abort is actually triggered via this condition. We argue first that the distributions of the latter two values are identical between $\mathcal{H}_2$ and $\mathcal{H}_1$, and then show how an adversary who avoids an abort when $e^{\mathsf{sk}} \neq 0$ can be used to solve the Computational Diffie-Hellman Problem.

Recall that in $\mathcal{H}_1$, per Step 5 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, $\mathcal{P}_h$ computes

$$\Gamma_h^2 = v_h \cdot \mathsf{pk} - w_h \cdot G \tag{40}$$

and that according to Step 4 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ we have

$$w_h = \mathsf{sk}_h \cdot v_h + \sum_{i \in \mathbf{P}^*} \left(w_h^{i,1} + w_h^{i,2} \right) \tag{41}$$

and that per Step 3 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ and the specification of $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, for $i \in \mathbf{P}^*$,

$$w_h^{i,1} + w_h^{i,2} = \mathsf{sk}_h \cdot (v_{*i} + e_i^v) + (\mathsf{sk}_{*i} + e_i^{\mathsf{sk}}) \cdot v_h - w_i^{h,1} - w_i^{h,2} \tag{42}$$

where sk_{*i} is $\mathcal{P}_i$'s additive share of sk as calculated from the known value of $p(i)$, v_{*i} is $\mathcal{P}_i$'s additive share of the adversary's output v_* from $\mathcal{F}^{n,t}_{\mathsf{Inv}}$, and e_i^v and e_i^{sk} are additive error terms that $\mathcal{P}_i$ can freely choose to include in their inputs to $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$. If we define

$$e^{\mathsf{sk}} = \sum_{i \in \mathbf{P}^*} e_i^{\mathsf{sk}} \qquad \text{and} \qquad e^v = \sum_{i \in \mathbf{P}^*} e_i^v$$

then via substitution of Equations 41 and 42 into Equation 40,

$$\begin{aligned}
\Gamma_h^2 &= v_h \cdot \mathsf{pk} - \left(\mathsf{sk}_h \cdot v_h + \sum_{i \in \mathbf{P}^*} \left(w_h^{i,1} + w_h^{i,2} \right) \right) \cdot G \\
&= \left(v_h \cdot \mathsf{sk}_* - \sum_{i \in \mathbf{P}^*} \left(w_h^{i,1} + w_h^{i,2} \right) \right) \cdot G \\
&= \left(\begin{array}{l} v_h \cdot \mathsf{sk}_* - \mathsf{sk}_h \cdot \sum_{i \in \mathbf{P}^*} (v_{*i} + e_i^v) \\ - v_h \cdot \sum_{i \in \mathbf{P}^*} \left(\mathsf{sk}_{*i} + e_i^{\mathsf{sk}} \right) + \sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2} \right) \end{array} \right) \cdot G \\
&= \left(\sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2} \right) - \mathsf{sk}_h \cdot (v_* + e^v) - v_h \cdot e^{\mathsf{sk}} \right) \cdot G
\end{aligned} \tag{43}$$

Observe next that in $\mathcal{H}_2$, per Step 5 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$,

$$\sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2} \right) = w_* - \mathsf{sk}_* \cdot v_* \tag{44}$$

Thus via substitution of Equation 44 into Equation 43, we have

$$\begin{aligned}
\Gamma_h^2 &= \left(w_* - \mathsf{sk}_* \cdot v_* - \mathsf{sk}_h \cdot (v_* + e^v) - v_h \cdot e^{\mathsf{sk}} \right) \cdot G \\
&= \left(w_* - \mathsf{sk}_h \cdot e^v - v_h \cdot e^{\mathsf{sk}} \right) \cdot G - v_* \cdot \mathsf{pk}
\end{aligned}$$

$$= \left(w_* + \mathsf{sk}_* \cdot e^v - v_h \cdot e^{\mathsf{sk}}\right) \cdot G - (v_* + e^v) \cdot \mathsf{pk} \tag{45}$$

which is exactly how Γ_h^2 is calculated in $\mathcal{H}_2$ per Step 7 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$. Thus Γ_h^2 is distributed identically in $\mathcal{H}_2$ and $\mathcal{H}_1$. Now consider the distribution of Γ^3 in $\mathcal{H}_1$. Specifically, recall that per Step 5 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$, $\mathcal{P}_h$ computes

$$\Gamma_h^3 = w_h \cdot R \tag{46}$$

which, via substitution of Equations 41 and 42 yields

$$\begin{aligned}
\Gamma_h^3 &= \left(\mathsf{sk}_h \cdot v_h + \sum_{i \in \mathbf{P}^*} \left(w_h^{i,1} + w_h^{i,2}\right)\right) \cdot R \\
&= \left(\begin{array}{l} \mathsf{sk}_h \cdot v_h + \mathsf{sk}_h \cdot \sum_{i \in \mathbf{P}^*} (v_{*i} + e_i^v) \\ + v_h \cdot \sum_{i \in \mathbf{P}^*} \left(\mathsf{sk}_{*i} + e_i^{\mathsf{sk}}\right) - \sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2}\right) \end{array}\right) \cdot R \\
&= \left(\begin{array}{l} \mathsf{sk}_h \cdot v_h + \mathsf{sk}_h \cdot (v_* + e^v) \\ + v_h \cdot \left(\mathsf{sk}_* + e^{\mathsf{sk}}\right) - \sum_{i \in \mathbf{P}^*} \left(w_i^{h,1} + w_i^{h,2}\right) \end{array}\right) \cdot R
\end{aligned} \tag{47}$$

Now recall that in $\mathcal{H}_2$, Equation 44 holds, and that per the specification of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$,

$$(v_h + v_*) \cdot R = G \tag{48}$$

and so by substitution of Equations 44 and 48 into Equation 47 we have

$$\begin{aligned}
\Gamma_h^3 &= \left(\mathsf{sk}_h \cdot v_h + \mathsf{sk}_h \cdot (v_* + e^v) + v_h \cdot \left(\mathsf{sk}_* + e^{\mathsf{sk}}\right) + \mathsf{sk}_* \cdot v_* - w_*\right) \cdot R \\
&= (\mathsf{sk}_h + \mathsf{sk}_*) \cdot G + \left(\mathsf{sk}_h \cdot e^v + v_h \cdot e^{\mathsf{sk}} - w_*\right) \cdot R \\
&= \mathsf{pk} + \left((\mathsf{sk} - \mathsf{sk}_*) \cdot e^v + v_h \cdot e^{\mathsf{sk}} - w_*\right) \cdot R \\
&= (1 + e^v \cdot (u_h + u_*)) \cdot \mathsf{pk} + \left(v_h \cdot e^{\mathsf{sk}} - \mathsf{sk}_* \cdot e^v - w_*\right) \cdot R
\end{aligned} \tag{49}$$

which is exactly how Γ_h^3 is calculated in $\mathcal{H}_2$ per Step 7 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$. Thus Γ_h^3 is also distributed identically in $\mathcal{H}_2$ and $\mathcal{H}_1$, and the two hybrids can be distinguished only by their distributions of aborts.

Consider the distinguishing case: that the adversary uses inputs in $\mathcal{H}_1$ such that it would be the case in $\mathcal{H}_2$ that $e^{\mathsf{sk}} \neq 0$, and yet avoids an abort. Avoidance of an abort implies that

$$\sum_{i \in \mathbf{P}^*} \Gamma_i^2 = -\Gamma_h^2$$

which, via substitution of Equation 45, yields

$$\sum_{i \in \mathbf{P}^*} \Gamma_i^2 = (v_* + e^v) \cdot \mathsf{pk} + \left(v_h \cdot e^{\mathsf{sk}} - w_* - \mathsf{sk}_* \cdot e^v\right) \cdot G$$

from which, by rearrangement, we derive

$$v_h \cdot G = \frac{1}{e^{\mathsf{sk}}} \left((w_* + \mathsf{sk}_* \cdot e^v) \cdot G - (v_* + e^v) \cdot \mathsf{pk} + \sum_{i \in \mathbf{P}^*} \Gamma_i^2 \right) \tag{50}$$

Notice that the right hand side is well defined and computable in $\mathcal{H}_2$, and recall that $R = G/(v_h + v_*)$ per the specification of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$. Thus, given a distinguisher for $\mathcal{H}_2$ and $\mathcal{H}_1$ and a uniform challenge $X = x \cdot G$, where x is unknown, we can generate the views of the corrupt parties as per $\mathcal{H}_2$, programming $R \coloneqq X + v_* \cdot G$, and then retrieve G/x via Equation 50. Doerner et al. [DKLs18] (Corollary F.2.1) show that if there exists an algorithm to compute G/x given a random challenge $X = x \cdot G$ which succeeds with probability ε, then there exists a reduction that solves the Computational Diffie-Hellman Problem with probability ε^6. Thus, under the Computational Diffie-Hellman Assumption, $\mathcal{H}_2$ is computationally indistinguishable from $\mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. Implement Step 7 of $\mathcal{S}_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}^*}$ completely. The only difference between this hybrid and $\mathcal{H}_2$ is that this hybrid aborts with certainty when $e^v \neq 0$, whereas $\mathcal{H}_2$ aborts according to the instructions in Step 7 of $\pi_{\mathsf{ECDSA\text{-}Sign}}^{n,t,\mathbf{P}}$; values in the view of the adversary are identical are identically distributed, even in the case of an abort.

Consider the distinguishing case: that the adversary uses inputs such that $e^v \neq 0$, and yet avoids an abort. Avoidance of an abort implies that

$$\sum_{i \in \mathbf{P}^*} \Gamma_i^3 = \mathsf{pk} - \Gamma_h^3$$

and by substitution of Equation 49, we have

$$\sum_{i \in \mathbf{P}^*} \Gamma_i^3 = \left(w_* + \mathsf{sk}_* \cdot e^v - v_h \cdot e^{\mathsf{sk}} \right) \cdot R - e^v \cdot (u_h + u_*) \cdot \mathsf{pk}$$

which, by rearrangement, yields

$$(u_h + u_*) \cdot \mathsf{pk} = \frac{1}{e^v} \cdot \left(\left(w_* + \mathsf{sk}_* \cdot e^v - v_h \cdot e^{\mathsf{sk}} \right) \cdot R - \sum_{i \in \mathbf{P}^*} \Gamma_i^3 \right) \tag{51}$$

Notice that the right hand side is well defined and computable in $\mathcal{H}_3$, and recall that $R = (u_h + u_*) \cdot G$ per the specification of $\mathcal{F}_{\mathsf{Inv}}^{n,t}$. Thus, given a distinguisher for $\mathcal{H}_3$ and $\mathcal{H}_2$ and a uniform challenge (X, Y) such that $X = x \cdot G$ and $Y = y \cdot G$ for unknown values of x and y, we can generate the corrupt parties' views as per $\mathcal{H}_2$, programming $\mathsf{pk} \coloneqq X$ and $R \coloneqq Y$, and then retrieve $x \cdot y \cdot G$ via Equation 51, solving the Computational Diffie-Hellman Problem for X and Y with no loss in probability relative to the success of the distinguisher. Thus, under the Computational Diffie-Hellman Assumption, $\mathcal{H}_3$ is computationally indistinguishable from $\mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. This hybrid implements the remaining instructions in $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ (i.e. Steps 8 through 11), replacing the instructions in Steps 5 through 9 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. Note that this hybrid differs from $\mathcal{H}_3$ only when the abort conditions in 7 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ are not triggered, and thus our argument considers this case only. The view of the adversary is characterized by the values Γ^2_h, Γ^3_h, and sig_h, as well as the abort condition in Step 9 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$. We know that when an abort is not triggered in Step 7 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$, $e^{\mathsf{sk}} = 0$ and $e^v = 0$.

Per Step 8 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$, we have in $\mathcal{H}_4$ that

$$\Gamma^2_h = w_* \cdot G - v_* \cdot \mathsf{pk} \qquad \text{and} \qquad \Gamma^3_h = \mathsf{pk} - w_* \cdot R$$

and via Steps 3 through 5 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA}}$, the specification of $\mathcal{F}^{n,t}_{\mathsf{Inv}}$ and $\mathcal{F}^{\ell}_{\mathsf{2PMul}}$, and the facts that $e^{\mathsf{sk}} = 0$ and $e^v = 0$, we know that

$$\begin{aligned} w_* &= (\mathsf{sk}_h + \mathsf{sk}_*) \cdot (v_h + v_*) - w_h \\ G &= (w_h + w_*) \cdot R \end{aligned}$$

and so, by substituting the latter two equations into the previous two, we have

$$\Gamma^2_h = v_h \cdot \mathsf{pk} - w_h \cdot G \qquad \text{and} \qquad \Gamma^3_h = w_h \cdot R$$

which is exactly the pair of equations used to calculate the check values in $\mathcal{H}_3$, per Step 5 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. Thus, these values are distributed identically in $\mathcal{H}_4$ and $\mathcal{H}_3$. Because the abort conditions in Step 7 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$ and Step 9 of $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ depend only upon these values, the distribution of aborts is identical between the two hybrids as well. Finally, we must consider the distribution of sig_h. In $\mathcal{H}_4$, per Step 11,

$$\mathsf{sig}_h = \mathsf{sig} - H(m) \cdot v_* - r_x \cdot w_*$$

We know that in $\mathcal{H}_3$, per Steps 8 and 9,

$$\mathsf{sig} = H(m) \cdot (v_h + v_*) + r_x \cdot (w_h + w_*)$$

and by substituting this into the previous equation we derive

$$\mathsf{sig}_h = H(m) \cdot v_i + r_x \cdot w_i$$

which is exactly how sig_h is computed in $\mathcal{H}_3$, per Step 9 of $\pi^{n,t,\mathbf{P}}_{\mathsf{ECDSA\text{-}Sign}}$. Thus $\mathcal{H}_4$ is distributed identically to $\mathcal{H}_3$.

Because $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Setup}}$ and $\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA\text{-}Sign}}$ are completely implemented, we have

$$\mathcal{H}_4 = \left\{ \mathsf{IDEAL}_{\mathcal{F}^{n,t}_{\mathsf{ECDSA}}, (\mathcal{S}^{n,t,\mathbf{P}^*}_{\mathsf{ECDSA}}, \mathcal{A}^{\mathbf{P}^*}), \mathcal{Z}}(z) \right\}_{\substack{z \in \{0,1\}^*, \\ \mathbf{P}^* \subset \mathbf{P} : |\mathbf{P}^*| = t-1}}$$

and by the foregoing sequence of hybrids, $\mathcal{H}_4 \overset{c}{\equiv} \mathcal{H}_0$; in other words, the view of a static adversary corrupting $t-1$ parties in the real world is computationally indistinguishable from a simulated execution of the same set of protocol instances under the Computational Diffie-Hellman Assumption in the $(\mathcal{F}^{n,t}_{\mathsf{Inv}}, \mathcal{F}^{\ell}_{\mathsf{2PMul}}, \mathcal{F}^{R_{\mathsf{DL}},n}_{\mathsf{Com\text{-}ZK}}, \mathcal{F}^{n}_{\mathsf{Com}})$-hybrid model, and Theorem 5.1 is proved. □