Secure Two-party Threshold ECDSA from ECDSA Assumptions

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*—The Elliptic Curve Digital Signature Algorithm (ECDSA) is one of the most widely used schemes in deployed cryptography. Through its applications in code and binary authentication, web security, and cryptocurrency, it is likely one of the few cryptographic algorithms encountered on a daily basis by the average person. However, its design is such that executing multi-party or threshold signatures in a secure manner is challenging: unlike other, less widespread signature schemes, secure multi-party ECDSA requires custom protocols, which has heretofore implied reliance upon additional cryptographic assumptions and primitives such as the Paillier cryptosystem.**

We propose new protocols for multi-party ECDSA key-generation and signing with a threshold of two, which we prove secure against malicious adversaries in the Random Oracle Model using only the Computational Diffie-Hellman Assumption and the assumptions already relied upon by ECDSA itself. Our scheme requires only two messages, and via implementation we find that it outperforms the best prior results in practice by a factor of 56 for key generation and 11 for signing, coming to within a factor of 18 of local signatures. Concretely, two parties can jointly sign a message in just over three milliseconds.

I. Introduction

Threshold Digital Signature Schemes, a classic notion in the field of Cryptography [2], 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 size. Though extensively studied, threshold signing is seldom used in practice, in part because threshold techniques for standard signatures tend to be highly inefficient, reliant upon unacceptable assumptions, or otherwise undesirable, while bespoke threshold schemes are incompatible with familiar and widely-accepted standards.

Consider the specific case of the Elliptic Curve Digital Signature Algorithm (ECDSA), perhaps the most widespread of signatures schemes: all existing threshold techniques for generating ECDSA signatures require the invocation of heavy cryptographic primitives such as Paillier encryption [3]–[5]. This leads both to poor performance and to reliance upon assumptions that are foreign to the mathematics on which ECDSA is based. This is troublesome, because performance concerns and avoidance of certain assumptions often motivate the use of ECDSA in the first place. We address this shortcoming by devising the first threshold signing algorithm for ECDSA that is based solely upon Elliptic Curves and the assumptions that the ECDSA signature scheme itself already makes. Furthermore, we improve upon the performance of previous works by a factor of eleven or more.

ECDSA is a standardized [6]–[8] derivative of the earlier Digital Signature Algorithm (DSA), devised by David Kravitz [9]. Where DSA is based upon arithmetic modulo a prime, ECDSA uses elliptic curve operations over finite fields. Compared to its predecessor, it has the advantage of being more efficient and requiring much shorter key lengths for the same level of security. In addition to the typical use cases of authenticated messaging, code and binary signing, remote login, &c., ECDSA has been eagerly adopted where high efficiency is important. For example, it is used by TLS [10], DNSSec [11], and many cryptocurrencies, including Bitcoin [12] and Ethereum [13].

A t-of-n threshold signature scheme is a set of protocols which allow n parties to jointly generate a single public key, along with n private shares of a joint secret key, and then privately sign messages if and only if t (some predetermined number) of those parties participate in the signing operation. In addition to satisfying the standard properties of signature schemes, it is necessary that threshold signature schemes be secure in a similar sense to other protocols for multi-party computation. That is, it is necessary that no malicious party can subvert the protocols to extract another party's share of the secret key, and that no subset of fewer than t parties can collude to generate signatures.

The concept of threshold signatures originates with the work of Yvo Desmedt [2], who proposed that multi-party and threshold cryptographic protocols could be designed to mirror societal structures, and thus cryptography could take on a new role, replacing organizational policy and social convention with mathematical assurance. Although this laid the motivational groundwork, it was the subsequent work of Desmedt and Frankel [14] that introduced the first true threshold encryption and signature schemes. These are based upon a combination of the well-known ElGamal [15] and Shamir Secret-Sharing [16] primitives, and carry the disadvantage that they require a trusted party to distribute private keys. Pedersen [17] later removed the need for a trusted third party.

The earliest threshold signature schemes were formulated as was convenient for achieving threshold properties; Desmedt and

This document revises and expands a paper that appeared in the 2018 IEEE S&P Conference under the same title (doi:10.1109/SP.2018.00036, [1]). This version differs in that the protocol has been modified and reorganized in order to simplify our security analysis at the cost of a slight performance penalty, and the appendix now contains a complete proof of security.

Frankel [14] recognized the difficulties inherent in designing threshold systems for standard signature schemes. Nevertheless, they later returned to the problem [18], proposing a non-interactive threshold system for RSA signatures [19]. This was subsequently improved and proven secure in a series of works [20]–[23]. Threshold schemes were also developed for Schnorr [24], [25] and DSA [26]–[28] signatures. Many of these schemes were too inefficient to be practical, however.

The efficiency and widespread acceptance of ECDSA make it a natural target for similar work, and indeed threshold ECDSA signatures are such a useful primitive that many cryptocurrencies are already implementing a similar concept in an ad-hoc manner [29]. Unfortunately, the design of the ECDSA algorithm poses a unique problem: the fact that it uses its nonce in a multiplicative fashion frustrates attempts to use typical linear secret sharing systems as primitives. The recent works of Gennaro *et al.* [4] and Lindell [3] solve this problem by using *multiplicative* sharing in combination with homomorphic Paillier encryption [30]; the former focuses on the general t-of-n threshold case, with an emphasis on the honest-majority setting, while the latter focuses on the difficult 2-of-2 case specifically. The resulting schemes (and the latter in particular) are very efficient in comparison to previous threshold schemes for plain DSA signatures: Lindell reports that his scheme requires only 37ms (including communication) per signature over the standard P-256 [8] curve.

Unfortunately, both Lindell and Gennaro *et al.*'s schemes depend upon the Paillier cryptosystem, and thus their security relies upon the Decisional Composite Residuosity Assumption. In some applications (cryptocurrencies, for example), the choice of ECDSA is made carefully in consideration of the required assumptions, and thus the use of a threshold scheme that requires new assumptions may not be acceptable. Additionally, if it is to be proven secure via simulation, Lindell's scheme requires a *new* (though reasonable) assumption about the Paillier cryptosystem to be made. Furthermore, the Paillier cryptosystem is so computationally expensive that even a single Paillier operation represents a significant cost relative to typical Elliptic Curve operations. Thus in this work we ask whether an efficient, secure, multi-party ECDSA signing scheme can be constructed using only elliptic curve primitives and elliptic curve assumptions, and find the answer in the affirmative.

A. Our Technique

Lindell observes that the problem of securely computing an ECDSA signature among two parties under a public key pk can be reduced to that of securely computing just *two* secure multiplications over the integers modulo the ECDSA curve order q. Lindell uses multiplicative shares of the secret key and nonce (hereafter called the *instance key*), and computes the signature using the Paillier additive homomorphic encryption scheme. We propose a new method to share the products which eliminates the need for homomorphic encryption.

Recall the signing equation for ECDSA,

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

where m is the message, H is a hash function, sk is the secret key, k is the instance key, and r_x is the x-coordinate of the elliptic curve point $R = k \cdot G$ (G being the generator for the curve). Suppose that $k = k_{\mathsf{A}} \cdot k_{\mathsf{B}}$ such that k_{A} and k_{B} are randomly chosen by Alice and Bob respectively, and $R = (k_{\mathsf{A}} \cdot k_{\mathsf{B}}) \cdot G$, and suppose that $\mathsf{sk} = \mathsf{sk}_{\mathsf{A}} \cdot \mathsf{sk}_{\mathsf{B}}$. Alice and Bob can learn R (and thus r_x) securely via Diffie-Hellman [31] exchange, and they receive m as input. Rearranging, we have

$$\mathsf{sig} = H(m) \cdot \left(\frac{1}{k_{\mathsf{A}}} \cdot \frac{1}{k_{\mathsf{B}}} \right) + r_x \cdot \left(\frac{\mathsf{sk}_{\mathsf{A}}}{k_{\mathsf{A}}} \cdot \frac{\mathsf{sk}_{\mathsf{B}}}{k_{\mathsf{B}}} \right)$$

which identifies the two multiplications on private inputs that are necessary. In our scheme, the results of of these multiplications are returned as *additive* secret shares to Alice and Bob. Since the rest of the equation is distributive over these shares, Alice and Bob can assemble shares of the signature without further interaction. Alice sends her share to Bob, who reconstructs sig and checks that it verifies.

To compute these multiplications, one could apply generic multi-party computation over arithmetic circuits, but generic MPC techniques incur large practical costs in order to achieve malicious security. Instead, we construct a new two-party multiplication protocol, based upon the semi-honest Oblivious-Transfer (OT) multiplication technique of Gilboa [32], which we harden to tolerate malicious adversaries. Note that even if the original Gilboa multiplication protocol is instantiated with a malicious-secure OT protocol, it is vulnerable to a simple selective failure attack whereby the OT sender (Alice) can learn one or more bits of the secret input of the OT receiver (Bob). We mitigate this attack by encoding the Bob's input randomly, such that Alice must learn more than a statistical security parameter number of bits in order to determine his unencoded input.

Unfortunately Bob may also cheat and learn something about Alice's secrets by using inconsistent inputs in the two different multiplication protocols, or by using inconsistent inputs between the multiplications and the Diffie-Hellman exchange. In order to mitigate this issue, we introduce a simple *consistency check* which ensures that Bob's inputs correspond to his shares of the established secret key and instance key. In essence, Alice and Bob combine their shares with the secret key and instance key *in the exponent*, such that if the shares are consistent then they evaluate to a constant value. This check is a novel and critical element of our protocol, and we conjecture that it can be applied to other domains.

Our signing protocol can easily be adapted for threshold signing among n parties with a threshold of two. This requires the addition of a special n-party setup protocol, and the modification of the signing protocol to allow the parties to provide additive shares of their joint secret key rather than multiplicative shares. Surprisingly, this incurs an overhead equivalent to less than half of an ordinary multiplication.

B. Our Contributions

1) We present an efficient n-party ECDSA key generation protocol and prove it secure in the Random Oracle Model

under the Computational Diffie-Hellman assumption.

2) We present an efficient two-party, two-round ECDSA signing protocol that is secure under the Computational Diffie-Hellman assumption and the assumption that the resulting signature is itself secure. Since CDH is implied by the Generic Group Model, under which ECDSA is proven secure, we require no additional assumptions relative to ECDSA itself.
3) We formulate a new ideal functionality for multi-party ECDSA signing that permits our signing protocol to achieve much better practical efficiency than it could if it were required to adhere to the standard functionality. We reduce the security of our functionality to the security of the classic signature game in the Generic Group Model.
4) In service of our main protocol, we devise a variant of Gilboa's multiplication by oblivious transfer technique [32] that may be of independent interest. It uses randomized input-encoding along with a new consistency check to maintain security against malicious adversaries.
5) At the core of our multiplication protocol is an oblivious transfer scheme based upon the Simplest OT [33] and KOS [34] OT-extension protocols. We introduce a new check system to avoid the issues that have recently cast doubt on the UC-security of Simplest OT [35].
6) We provide an implementation of our protocol in Rust, and demonstrate its efficiency under real-world conditions. We find our implementation can produce roughly 320 signatures per second per core on commodity hardware.

C. Organization

The remainder of this document is organized as follows. In Section II we review essential concepts and definitions, and in Section III we discuss the ideal functionality that our protocols will realize. In Section IV we specify a basic two-party protocol, which we extend to support 2-of-n threshold signing in Section V. In Section VI we describe the multiplication primitive that we use. In Section VII we present a comparative analysis of our protocols. In Section VIII, we describe our implementation and present benchmark results. In the appendices we describe our OT primitive, further discuss the functionalities we use, and prove our protocols secure.

II. Preliminaries and Definitions

A. Notation and Conventions

We denote curve points with capitalized variables and scalars with lower case. Vectors are given in bold and indexed by subscripts, while matrices are denoted by bold capitals, with subscripts and superscripts representing row indices and column indices respectively. Thus $\mathbf{x}_i$ is the i^{th} element of the vector $\mathbf{x}$, which is distinct from the variable x. We use $=$ to denote equality, $:=$ for assignment, $\leftarrow$ for sampling an instance from a distribution, and $\overset{c}{\equiv}$ to indicate computational indistinguishability for two distributions. The i^{th} party participating in a protocol is denoted with $\mathcal{P}_i$, and when only two parties participate, they are called Alice and Bob for convenience. Throughout this document, we use κ to represent the security parameter of the elliptic curve over which our equations are evaluated, and we use s for a statistical security parameter.

In functionalities, we assume standard and implicit bookkeeping. 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, and we assume that the functionality aborts if a party tries to reuse a session ID, send messages out of order, &c. 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. So, for example, many of our functionalities are parameterized by a group $\mathbb{G}$ of order q, but we leave implicit the fact that in any given instantiation all functionalities use the same group.

Finally, we use H throughout this document to denote a hash function, which is modeled as a random oracle. It takes the form $H^n : \{0,1\}^* \mapsto \mathbb{Z}_q^n$. That is, the range of the function is n elements from $\mathbb{Z}_q$, where n is given as a superscript, and assumed to be 1 if absent. If a subscript is present in a call to H, then the function returns only the element from its output that is indexed by the subscript. Thus, for example, $H^2(x) = (H^2_1(x), H^2_2(x))$.

B. Digital Signatures

Definition 1 (Digital Signature Scheme [36])**.**
A *Digital Signature Scheme* is a tuple of probabilistic polynomial time (PPT) 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 or invalid: $b := \mathsf{Verify}_{\mathsf{pk}}(m, \sigma)$

A Digital Signature Scheme satisfies two properties:

1) (Correctness) With overwhelmingly high probability, all valid signatures must verify. Formally, we require that over $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and all messages m in the message space,
$$\Pr_{\mathsf{pk},\mathsf{sk},m}\Big[\mathsf{Verify}_{\mathsf{pk}}(m, \mathsf{Sign}_{\mathsf{sk}}(m)) = 1\Big] > 1 - \mathrm{negl}(\kappa)$$
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,
$$\Pr_{\mathsf{pk},\mathsf{sk}}\left[\begin{array}{r}\mathsf{Verify}_{\mathsf{pk}}(m, \sigma) = 1 \wedge m \notin \mathbf{Q} : \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})\end{array}\right] < \mathrm{negl}(\kappa)$$

C. ECDSA

The ECDSA algorithm 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. The algorithm makes use of the hash function H. Curve coordinates and scalars are represented in $\kappa = |q|$ bits, which is also the security parameter. A number of standard curves with various security parameters have been promulgated [8]. Assuming a curve has been fixed, the ECDSA algorithms are as follows [36]:

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

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 $(r'_x \mod q) = (r_x \mod q)$.

The initial publication of the ECDSA algorithm did not include a rigorous proof of security; this proof was later provided by Brown [37] in the Generic Group Model, based upon the hardness of discrete logarithms and the assumption that the hash function H is collision resistant and uniform. Vaudenay [38] surveys this and other ECDSA security results, and Koblitz and Menezes provide some analysis and critique of the proof technique [39]. In this work, we simply assume that ECDSA is secure as specified in Definition 1.

D. Oblivious Transfer

Our construction uses a 1-of-2 Oblivious Transfer (OT) system, which is a cryptographic protocol evaluated by two parties: a sender and a receiver. The sender submits as input two private messages, m_0 and m_1; the receiver submits a single bit b, indicating its choice between those two. At the end of the protocol, the receiver learns the message m_b, and the sender learns nothing. In particular, the sender does not learn the value of the bit b, and the receiver does not learn the value of the message $m_{\bar{b}}$. 1-of-2 OT was introduced by Evan *et al.* [40], and is distinct from the earlier Rabin-style OT [41], [42]. For a complete formal definition, we refer the reader to Naor and Pinkas [43]. Beaver [44] later introduced the notion of OT-extension, by which a few instances of Oblivious Transfer can be extended to transfer polynomially many messages using only symmetric-key primitives. For reasons of efficiency, many modern protocols use OT-extension rather than plain OT.

III. Two Functionalities

As our scheme is a multi-party computation protocol in the malicious security model, its security will be defined relative to an ideal functionality. Prior works on threshold ECDSA [3], [4] present a functionality $\mathcal{F}_{\mathsf{ECDSA}}$ that applies the threshold model directly to the original ECDSA algorithms. The ECDSA Gen algorithm becomes the first phase of $\mathcal{F}_{\mathsf{ECDSA}}$, and the ECDSA Sign algorithm becomes the second.

Functionality 1. $\mathcal{F}_{\mathsf{ECDSA}}$**:**

This functionality is parameterized by a group $\mathbb{G}$ of order q (represented in κ bits) generated by G, as well as hash function H. The setup phase runs once with a group of parties $\{\mathcal{P}_i\}_{i\in[1,n]}$, and the signing phase may be run many times between any two specific parties from this group. For convenience, we refer to these two parties as Alice and Bob.

Setup (2-of-*n*): On receiving $(\texttt{init})$ 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 $(\texttt{public-key}, \mathsf{pk})$ to all parties.
4) Store $(\texttt{ready})$ in memory.

Signing: On receiving $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, \mathsf{B}, m)$ from Alice and $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, \mathsf{A}, m)$ from Bob, if $(\texttt{ready})$ exists in memory but $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ does not exist in memory:

1) Sample $k \leftarrow \mathbb{Z}_q$ and store it as the instance key.
2) Compute $(r_x, r_y) = R := k \cdot G$.
3) Compute
$$\mathsf{sig} := \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
4) Collect the signature, $\sigma := (\mathsf{sig} \mod q, r_x \mod q)$.
5) Send $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \sigma)$ to Bob.
6) Store $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ in memory.

Our scheme does not realize $\mathcal{F}_{\mathsf{ECDSA}}$, but instead a new functionality $\mathcal{F}_{\mathsf{SampledECDSA}}$, which we have formulated to allow us to build a protocol that requires only two rounds. It is well known that generic Multi-party Computation can compute *any* function in two rounds [45], [46] (or even one round, with a complex setup procedure), but the challenge is to do so efficiently. It is natural to use a Diffie-Hellman exchange to compute R, which would otherwise require expensive secure point multiplication techniques, but this precludes either a two-round protocol or use of the standard functionality for an intuitive reason: in the (basic) Diffie-Hellman exchange, Bob sends $D_{\mathsf{B}} := k_{\mathsf{B}} \cdot G$ to Alice, who replies to Bob with $D_{\mathsf{A}} := k_{\mathsf{A}} \cdot G$. Both Alice and Bob can compute $R := k_{\mathsf{A}} \cdot k_{\mathsf{B}} \cdot G$. While Alice cannot learn the discrete logarithm of R, she does have the power to determine R itself due to the fact that she chooses k_{A} after having seen D_{B}. This conflicts with $\mathcal{F}_{\mathsf{ECDSA}}$, which requires that the functionality pick R. It is not obvious how to solve this without adding rounds or using a much more expensive primitive, though we conjecture that a more elaborate one-time setup procedure may provide a resolution.

Instead, we have devised $\mathcal{F}_{\mathsf{SampledECDSA}}$. Relative to $\mathcal{F}_{\mathsf{ECDSA}}$, we divide the signing phase of the functionality into three parts, allowing the parties to abort between them. In the first two parts, Alice and Bob initiate a new signature for a message m, and a random instance key k is chosen by the functionality, along with $R = k \cdot G$, which is returned to Alice. Alice is permitted to request a new sampling of R from the functionality arbitrarily many times (with a negligible chance of receiving a favorable value), and to choose from the sampled set one value under which the signature will be performed. If neither party aborts, then in the third part the functionality will return a signature under the chosen R. This accounts for Alice's ability to manipulate the Diffie-Hellman exchange, and yet it ensures that she does not know the discrete logarithm of the value that is eventually chosen, and that the value is uniform over $\mathbb{G}$.

In Appendix C we prove in the Generic Group Model [47] that $\mathcal{F}_{\mathsf{SampledECDSA}}$ is no less secure than ECDSA itself. We also believe that a four-round variant of our protocol can realize the $\mathcal{F}_{\mathsf{ECDSA}}$ functionality directly.

Functionality 2. $\mathcal{F}_{\mathsf{SampledECDSA}}$**:**

This functionality is parametrized in a manner identical to $\mathcal{F}_{\mathsf{ECDSA}}$. Note that Alice may engage in the Offset Determination phase as many times as she wishes.

Setup (2-of-n): On receiving $(\texttt{init})$ 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 $(\texttt{public-key}, \mathsf{pk})$ to all parties.
4) Store $(\texttt{ready})$ in memory.

Instance Key Agreement: On receiving $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{B})$ from Alice and $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{A})$ from Bob, if $(\texttt{ready})$ exists in memory, and if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, \cdot, \cdot)$ does not exist in memory, and if Alice and Bob both supply the same message m and each indicate the other as their counterparty, then:

1) Sample $k_{\mathsf{B}} \leftarrow \mathbb{Z}_q$.
2) Store $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ in memory.
3) Send $(\texttt{nonce-shard}, \mathsf{id}^{\mathsf{sig}}, D_{\mathsf{B}} := k_{\mathsf{B}} \cdot G)$ to Alice.

Offset Determination: On receiving $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i)$ from Alice, if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ exists in memory, but $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, j, \cdot)$ for $j = i$ does not exist in memory:

4) Sample $k_i^{\Delta} \leftarrow \mathbb{Z}_q$.
5) Store $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i, k_i^{\Delta})$ in memory.
6) Compute $k_{i,\mathsf{A}}^{\Delta} = k_i^{\Delta}/k_{\mathsf{B}}$ and send $(\texttt{offset}, \mathsf{id}^{\mathsf{sig}}, k_{i,\mathsf{A}}^{\Delta})$ to Alice.

Signing: On receiving $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, i, k_{\mathsf{A}})$ from Alice and $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}})$ from Bob, if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ exists in memory and $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, j, R_i, k_i^{\Delta})$ for $j = i$ exists in memory, but $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ does not exist in memory:

7) Abort if $k_{\mathsf{A}} \cdot k_{\mathsf{B}} \cdot G \neq R_i$.
8) Set $k := k_{\mathsf{A}} \cdot k_{\mathsf{B}} + k_i^{\Delta}$ and store $(r_x, r_y) = R := k \cdot G$.
9) Compute
$$\mathsf{sig} := \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
10) Collect the signature, $\sigma := (\mathsf{sig} \mod q, r_x \mod q)$.
11) Send $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, R, k_i^{\Delta}, \sigma)$ to Bob.
12) Store $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ in memory.

IV. A Basic 2-of-2 Scheme

We describe a simplified 2-of-2 version of our scheme initially, abstracting away the multiplication protocols for the sake of clarity. In Section V we extend our scheme to support 2-of-n threshold signing. The fundamental structure of our 2-of-2 scheme is similar to that of Lindell [3] in that the signing protocol ingests multiplicative shares of both the private key and the instance key from each party.

A. Signing

Alice and Bob begin with m, the message to be signed, and multiplicative shares of a secret key (sk_{A} and sk_{B} respectively), as well as a public key pk that is consistent with those shares. The protocol is divided into four logical steps:

1) **Multiplication:** The parties transform multiplicative shares of the instance key into additive shares. A second multiplication converts multiplicative shares of the secret key divided by the instance key into additive shares. Due to the presence of the consistency check and verification steps (below), the multiplication protocols employed are not required to enforce correctness or consistency of inputs; thus we model multiplication via $\mathcal{F}_{\mathsf{Mul}}$ (given in Section VI), which allows for well-specified cheating. To instantiate this functionality, we use the custom OT-based multiplication protocol that we describe in Section VI-B.
2) **Instance Key Exchange:** The parties calculate $R = k \cdot G$ using a modified Diffie-Hellman exchange.
3) **Consistency Check:** The parties verify that the first multiplication uses inputs consistent with the Instance Key Exchange. This is achieved by adding a random pad ϕ to Alice's input, and then combining the pad with the multiplication output and the known value R in such a way that Bob can retrieve the pad only if he acted honestly. A second check ensures that the multiplications are consistent with each other and with the public key, by combining the multiplication outputs with the public key in the exponent.
4) **Signature and Verification:** The parties reconstruct the signature, which is given to Bob. If the signature verifies in the usual way, then Bob outputs it.

The Instance Key Exchange component implements the second and third phases of the $\mathcal{F}_{\mathsf{SampledECDSA}}$ functionality, and the Multiplication, Consistency Check, and Verification components implement the fourth phase. Although we make a logical distinction between these four components, in the actual protocol they are intertwined. In particular, we reorder the messages such that all messages from Bob to Alice come first, followed by all messages from Alice to Bob, which results in a two-message protocol. Additionally, rather than perform the consistency check directly, we use its associated value as a key to encrypt all subsequent communications, so that the protocol can only be completed if the consistency check passes.

A proof of knowledge is necessary in order to ensure that Alice's inputs are extractable, and thus the protocol makes use of a zero-knowledge proof-of-knowledge-of-discrete-logarithm functionality $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, which is specified in Appendix B. This can be concretely instantiated by a Schnorr proof [24] and the Fiat-Shamir [48] or Fischlin [49] transform. We give the signing protocol below, and in Figure 1 we provide an illustration, along with annotations indicating the logical component associated with each step.

Protocol 1. Two-party Signing $(\pi_{\mathsf{2P\text{-}ECDSA}}^{\mathsf{Sign}})$**:**

This protocol is parameterized by the Elliptic curve $(\mathbb{G}, G, q)$ and the hash function H. It relies upon the $\mathcal{F}_{\mathsf{Mul}}$ and $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ functionalities. Alice and Bob provide their multiplicative secret key shares $\mathsf{sk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}$ as input, along with identical copies of the message m, and Bob receives as output a signature σ.

Multiplication and Instance Key Exchange:

1) Bob chooses his secret instance key, $k_\mathsf{B} \leftarrow \mathbb{Z}_q$, and Alice chooses her instance key seed, $k'_\mathsf{A} \leftarrow \mathbb{Z}_q$. Bob computes $D_\mathsf{B} := k_\mathsf{B} \cdot G$ and sends D_B to Alice.
2) Alice computes
$$R' := k'_\mathsf{A} \cdot D_\mathsf{B}$$
$$k_\mathsf{A} := H(R') + k'_\mathsf{A}$$
$$R := k_\mathsf{A} \cdot D_\mathsf{B}$$
3) Alice chooses a pad $\phi \leftarrow \mathbb{Z}_q$, and then Alice and Bob invoke the $\mathcal{F}_{\mathsf{Mul}}$ functionality with inputs $\phi + 1/k_\mathsf{A}$ and $1/k_\mathsf{B}$ respectively, and receive shares t^1_A and t^1_B of their padded joint inverse instance key
$$t^1_\mathsf{A} + t^1_\mathsf{B} = \frac{\phi}{k_\mathsf{B}} + \frac{1}{k_\mathsf{A} \cdot k_\mathsf{B}}$$
Alice and Bob also invoke $\mathcal{F}_{\mathsf{Mul}}$ with inputs $\mathsf{sk}_\mathsf{A}/k_\mathsf{A}$ and $\mathsf{sk}_\mathsf{B}/k_\mathsf{B}$. They receive shares t^2_A and t^2_B of their joint secret key over their joint instance key
$$t^2_\mathsf{A} + t^2_\mathsf{B} = \frac{\mathsf{sk}_\mathsf{A} \cdot \mathsf{sk}_\mathsf{B}}{k_\mathsf{A} \cdot k_\mathsf{B}}$$
The protocol instances that instantiate $\mathcal{F}_{\mathsf{Mul}}$ are interleaved such that the messages from Bob to Alice are transmitted first, followed by Alice's replies.
4) Alice transmits R' to Bob, who computes
$$R := H(R') \cdot D_\mathsf{B} + R'$$
For both Alice and Bob let $(r_x, r_y) = R$.
5) Alice submits $(\texttt{prove}, k_\mathsf{A}, D_\mathsf{B})$ to $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, and Bob submits $(\texttt{prove}, R, D_\mathsf{B})$. Bob receives a bit indicating whether the proof was sound. If it was not, he aborts.

Consistency Check, Signature, and Verification:

6) Alice and Bob both compute $m' = H(m)$.
7) Alice computes the first check value Γ^1, encrypts her pad ϕ with Γ^1, and transmits the encryption η^ϕ to Bob.
$$\Gamma^1 := G + \phi \cdot k_\mathsf{A} \cdot G - t^1_\mathsf{A} \cdot R$$
$$\eta^\phi := H(\Gamma^1) + \phi$$
8) Alice computes her share of the signature sig_A and the second check value Γ^2. She encrypts sig_A with Γ^2 and then transmits the encryption η^{sig} to Bob
$$\mathsf{sig}_\mathsf{A} := (m' \cdot t^1_\mathsf{A}) + (r_x \cdot t^2_\mathsf{A})$$
$$\Gamma^2 := (t^1_\mathsf{A} \cdot \mathsf{pk}) - (t^2_\mathsf{A} \cdot G)$$
$$\eta^{\mathsf{sig}} := H(\Gamma^2) + \mathsf{sig}_\mathsf{A}$$
9) Bob computes the check values and reconstructs the signature
$$\Gamma^1 := t^1_\mathsf{B} \cdot R$$
$$\phi := \eta^\phi - H(\Gamma^1)$$
$$\theta := t^1_\mathsf{B} - \phi/k_\mathsf{B}$$
$$\mathsf{sig}_\mathsf{B} := (m' \cdot \theta) + (r_x \cdot t^2_\mathsf{B})$$
$$\Gamma^2 := (t^2_\mathsf{B} \cdot G) - (\theta \cdot \mathsf{pk})$$
$$\mathsf{sig} := \mathsf{sig}_\mathsf{B} + \eta^{\mathsf{sig}} - H(\Gamma^2)$$
10) Bob uses the public key pk to verify that $\sigma := (\mathsf{sig}, r_x)$ is a valid signature on message m. If the verification fails, Bob aborts. If it succeeds, he outputs σ.

On the Structure of the Consistency Check: Because the consistency check mechanism is non-obvious, we present an informal justification for it here. In Appendix F, we prove the mechanism formally secure. Suppose that we reorganized our protocol to omit Alice's pad ϕ. Then we would have
$$t^1_\mathsf{A} + t^1_\mathsf{B} = \frac{1}{k_\mathsf{A} \cdot k_\mathsf{B}} \qquad t^2_\mathsf{A} + t^2_\mathsf{B} = \frac{\mathsf{sk}_\mathsf{A} \cdot \mathsf{sk}_\mathsf{B}}{k_\mathsf{A} \cdot k_\mathsf{B}}$$
$$(t^1_\mathsf{A} + t^1_\mathsf{B}) \cdot \mathsf{pk} = (t^2_\mathsf{A} + t^2_\mathsf{B}) \cdot G$$

If Bob behaves honestly, he should use $1/k_\mathsf{B}$ and $\mathsf{sk}_\mathsf{B}/k_\mathsf{B}$ as his inputs to the two multiplications. Suppose Bob cheats by using different inputs; without loss of generality, we can interpret his cheating as using inputs $x + 1/k_\mathsf{B}$ and $\mathsf{sk}_\mathsf{B}/k_\mathsf{B}$, in essence offsetting his input for the first multiplication by some value x relative to his input for the second multiplication:
$$t^1_\mathsf{A} + t^1_\mathsf{B} = 1/k + x/k_\mathsf{A}$$
$$(t^1_\mathsf{A} + t^1_\mathsf{B}) \cdot \mathsf{pk} = (t^2_\mathsf{A} + t^2_\mathsf{B}) \cdot G + x \cdot \mathsf{pk}/k_\mathsf{A}$$
To pass the consistency check, Bob would need to calculate pk/k_A, which we show by means of a reduction is as hard as breaking the Computational Diffie-Hellman problem.

In our hypothetical scenario, it is tempting to take advantage of the fact that $(t^1_\mathsf{A} + t^1_\mathsf{B}) \cdot R = G$ to design a similar mechanism to verify that the first multiplication is consistent with the instance key exchange, but a check based upon this principle is insecure. Again, if we suppose that Bob cheats by offsetting his input for the multiplication by some value x relative to his input for the Diffie-Hellman exchange that produces R, then
$$t^1_\mathsf{A} + t^1_\mathsf{B} = 1/k + x/k_\mathsf{A}$$
$$(t^1_\mathsf{A} + t^1_\mathsf{B}) \cdot R = G + x \cdot k_\mathsf{B} \cdot G$$
Unfortunately, the offset produced is made up entirely of elements known to Bob. We rectify this by introducing into the

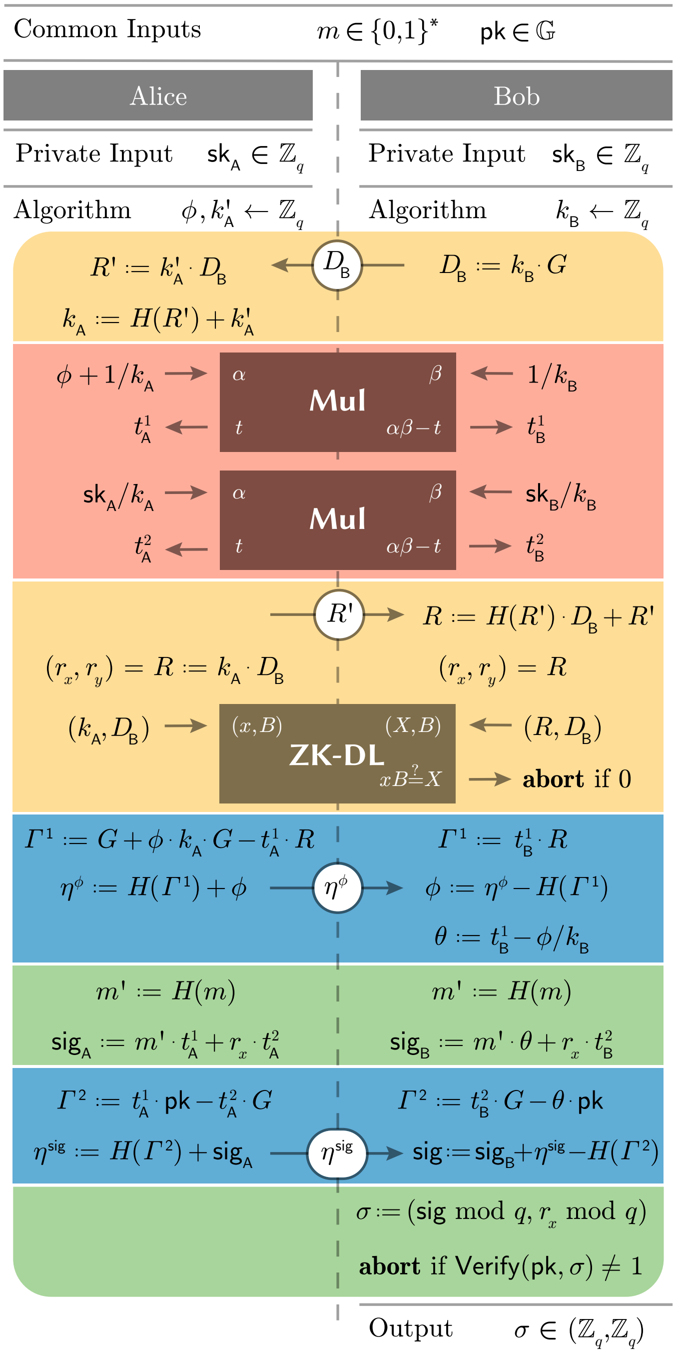

Fig. 1: **Illustrated Two-party Signing Scheme**. Operations are color-coded according to the logical component with which they are associated: Multiplication, Instance Key Exchange, Consistency Check, and Verification/Signing. We specify how to instantiate the multiplication subprotocol (π_{Mul}) in Section VI-B.

equation a term that Bob cannot predict. Alice intentionally offsets her input to the multiplication using a pad ϕ, giving us the system presented in $\pi^{\mathsf{Sign}}_{\mathsf{2P\text{-}ECDSA}}$. If Bob is honest, then

$$t^1_\mathsf{A} + t^1_\mathsf{B} = 1/k + \phi/k_\mathsf{B}$$
$$t^1_\mathsf{B} \cdot R = G + \phi \cdot k_\mathsf{A} \cdot G - t^1_\mathsf{A} \cdot R$$

which implies that both Alice and Bob can compute $t^1_\mathsf{B} \cdot R$. On the other hand, if Bob is dishonest, then

$$t^1_\mathsf{A} + t^1_\mathsf{B} = 1/k + \phi/k_\mathsf{B} + x/k_\mathsf{A} + x \cdot \phi$$
$$t^1_\mathsf{B} \cdot R = G + \phi \cdot k_\mathsf{A} \cdot G + x \cdot k_\mathsf{B} \cdot G + x \cdot \phi \cdot R - t^1_\mathsf{A} \cdot R$$

Because x is unknown to Alice and ϕ is unknown to Bob, neither party is capable of calculating the offset that has been induced. Consequently, if Alice masks ϕ using the value of $t^1_\mathsf{B} \cdot R$ that she *expects* Bob to have, then he will be able to remove the mask and retrieve ϕ if and only if he has behaved honestly. Without knowledge of ϕ, he will not be able to pass the second consistency check or reconstruct the signature. We note that there is an assumption of circular security in this construction, which we resolve via the Random Oracle Model.

B. Setup

We now present a simplified setup protocol for two parties. This protocol does not implement the setup phase of the $\mathcal{F}_{\mathsf{SampledECDSA}}$ functionality, as it does not support threshold signing (we extend it to do so in Section V), but it does provide a similar functionality to the setup protocol of Lindell [3]. In short, it implements the ECDSA Gen algorithm, combining multiplicative secret key shares via a simple Diffie-Hellman key exchange. Proofs of knowledge are necessary in order to ensure that if the protocol completes then the parties are capable of signing; in addition to the $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$ functionality, this protocol makes use of a commit-and-prove variant $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, which is specified in Appendix B. Finally, the parties notify the $\mathcal{F}_{\mathsf{Mul}}$ functionality that they are ready, which corresponds to the initialization of OT-extensions.

Protocol 2. Two-party Setup $(\pi^{\mathsf{Setup}}_{\mathsf{2P\text{-}ECDSA}})$**:**

This protocol is parameterized by the Elliptic curve $(\mathbb{G}, G, q)$, and relies upon the $\mathcal{F}_{\mathsf{Mul}}$, $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$, and $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ functionalities. It takes no input and yields the secret key shares sk_A and sk_B to Alice and Bob respectively, along with the joint public key pk to both parties.

Public Key Generation:

1) Alice and Bob sample $\mathsf{sk}_\mathsf{A} \leftarrow \mathbb{Z}_q$ and $\mathsf{sk}_\mathsf{B} \leftarrow \mathbb{Z}_q$, respectively, and then they compute $\mathsf{pk}_\mathsf{A} := \mathsf{sk}_\mathsf{A} \cdot G$ and $\mathsf{pk}_\mathsf{B} := \mathsf{sk}_\mathsf{B} \cdot G$.
2) Alice submits $(\texttt{com-proof}, \mathsf{sk}_\mathsf{A}, G)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, and Bob becomes aware of Alice's commitment.
3) Bob sends pk_B to Alice and submits $(\texttt{prove}, \mathsf{sk}_\mathsf{B}, G)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$. Alice submits $(\texttt{prove}, \mathsf{pk}_\mathsf{B}, G)$, and receives a bit indicating whether the proof was sound. If it was not, she aborts.
4) Alice sends pk_A to Bob and instructs $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ to release the proof associated with her previous commitment. Bob submits $(\texttt{prove}, \mathsf{pk}_\mathsf{A}, G)$, and receives a bit indicating whether the proof was sound. If it was not, he aborts.
5) Alice and Bob compute $\mathsf{pk} := \mathsf{sk}_\mathsf{A} \cdot \mathsf{pk}_\mathsf{B} = \mathsf{sk}_\mathsf{B} \cdot \mathsf{pk}_\mathsf{A}$.

Auxiliary Setup:

6) Alice and Bob both send the $(\texttt{init})$ messages to the $\mathcal{F}_{\mathsf{Mul}}$ Functionality to initialize OT-extensions.

V. 2-OF-n THRESHOLD SIGNING

We now demonstrate a simple extension of our two-party ECDSA protocol for performing threshold signatures among n parties, with a threshold of two. With this extension, our protocol realizes the $\mathcal{F}_{\mathsf{SampledECDSA}}$ functionality that we gave in Section III. In $\pi^{\mathsf{Setup}}_{\mathsf{2P\text{-}ECDSA}}$, Alice and Bob supplied multiplicative shares $\mathsf{sk_A}$ and $\mathsf{sk_B}$ of their joint secret key. In the threshold setting we will be working with a set of parties $\mathcal{P}$ of size n, each party i with a secret key share sk_i, and we demand that if the setup does not abort then any pair of parties can sign. In order to achieve this, we specify that in the threshold setting, the joint secret key sk is calculated as the *sum* of the parties' contributions, rather than as the product:

$$\mathsf{sk} := \sum_{i\in[1,n]} \mathsf{sk}_i$$

In other words, the parties' individual secret keys represent an n-of-n sharing of sk. It is natural to use a threshold secret sharing scheme to convert these into a 2-of-n sharing. Specifically, we use Shamir Secret Sharing [16], and a simple consistency check allows us to guarantee security against malicious adversaries.

From Shamir shares, any two parties can generate additive shares of the joint secret key. However, our 2-of-2 signing protocol ($\pi^{\mathsf{Sign}}_{\mathsf{2P\text{-}ECDSA}}$) required multiplicative shares as its input. We will need to modify the signing protocol slightly to account for the change. First, we present our 2-of-n setup procedure.

A. Setup

Protocol 3. 2-of-n Setup $\left(\pi^{\mathsf{2P\text{-}Setup}}_{n\mathsf{P\text{-}ECDSA}}\right)$:

This protocol is parameterized by the Elliptic curve $(\mathbb{G}, G, q)$, and relies $\mathcal{F}_{\mathsf{Mul}}$ and $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ functionalities. It runs among a group of parties $\mathcal{P}$ of size n, taking no input, and yielding to each party $\mathcal{P}_i$ a point $p(i)$ on the polynomial p, a secret key share sk_i, and the joint public key pk.

Public Key Generation:

1) For all $i \in [1, n]$, Party $\mathcal{P}_i$ samples $\mathsf{sk}_i \leftarrow \mathbb{Z}_q$.
2) For all $i \in [1, n]$, Party $\mathcal{P}_i$ calculates $\mathsf{pk}_i := \mathsf{sk}_i \cdot G$ and submits $(\texttt{com-proof}, \mathsf{sk}_i, G)$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$, which notifies the other parties that $\mathcal{P}_i$ is committed. When $\mathcal{P}_i$ becomes aware of all other parties' commitments, it sends pk_i to the other parties and instructs $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}}$ to release its proof to them. All other parties submit $(\texttt{prove}, \mathsf{pk}_i, G)$ and receive a bit indicating whether the proof was sound. If any party's proof fails to verify, then all parties abort.
3) All parties compute the shared public key
$$\mathsf{pk} := \sum_{i\in[1,n]} \mathsf{pk}_i$$
4) For all $i \in [1, n]$, $\mathcal{P}_i$ chooses a random line given by the degree-1 polynomial $p_i(x)$, such that $p_i(0) = \mathsf{sk}_i$. For all $j \in [1, n]$, $\mathcal{P}_i$ sends $p_i(j)$ to $\mathcal{P}_j$ and receives $p_j(i)$.
5) For all $i \in [1, n]$, $\mathcal{P}_i$ computes its point
$$p(i) := \sum_{j\in[1,n]} p_j(i)$$
It also computes a commitment to its share of the secret key, $T_i := p(i) \cdot G$, and broadcasts T_i to all other parties.
6) All parties abort if there exists $i \in [2, n]$ such that
$$\lambda_{(i-1),i} \cdot T_{i-1} + \lambda_{i,(i-1)} \cdot T_i \neq \mathsf{pk}$$
where $\lambda_{(i-1),i}$ and $\lambda_{i,(i-1)}$ are the appropriate Lagrange coefficients for Shamir-reconstruction between $\mathcal{P}_{i-1}$ and $\mathcal{P}_i$. If any party holds a point $p(i)$ that is inconsistent with the polynomial held by the other parties, then this check will fail.

Auxilliary Setup:

7) Every pair of parties $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $i < j$ send the $(\texttt{init})$ message to the $\mathcal{F}_{\mathsf{Mul}}$ functionality.

A Note on General Thresholds: We note that a slight generalization of the $\pi^{\mathsf{2P\text{-}Setup}}_{n\mathsf{P\text{-}ECDSA}}$ protocol allows it to perform setup for any threshold t such that $t \leq n$. The only required changes are the use of polynomials of the appropriate degree (as in Shamir Secret Sharing), and the evaluation of the consistency check in Step 6 over contiguous threshold-sized groups of parties. However, our signing protocol is not so easily generalized; therefore we leave general threshold signing to future work, and focus here on the 2-of-n case.

B. Signing

Once the setup is complete, suppose two parties from the set $\mathcal{P}$ (we will resume referring to them as Alice and Bob) wish to sign. They can use Lagrange interpolation [50] to construct additive shares t^0_{A} and t^0_{B} of the secret key, but the signing algorithm we have previously described requires *multiplicative* shares. To account for this, we modify our signing algorithm in the following intuitive way: originally, the second invocation of $\mathcal{F}_{\mathsf{Mul}}$ took $\mathsf{sk_A}/k_{\mathsf{A}}$ from Alice and $\mathsf{sk_B}/k_{\mathsf{B}}$ from Bob and computed additive shares of the product

$$\frac{\mathsf{sk_A} \cdot \mathsf{sk_B}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}}$$

We replace this with two invocations of $\mathcal{F}_{\mathsf{Mul}}$ that calculate

$$\frac{t^0_{\mathsf{A}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} \quad \text{and} \quad \frac{t^0_{\mathsf{B}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}}$$

respectively. Alice and Bob can then locally sum their outputs from these two multiplications to yield shares of

$$\frac{t^0_{\mathsf{A}} + t^0_{\mathsf{B}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} = \frac{\mathsf{sk}}{k}$$

What follows is our 2-of-n signing protocol, in its entirety. We note that the Consistency Check, Signature, and Verification phase of this protocol is identical to the corresponding phase in the $\pi^{\mathsf{Sign}}_{\mathsf{2P\text{-}ECDSA}}$ protocol that we gave in Section IV-A.

Protocol 4. 2-of-n Signing $\left(\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}\right)$**:**

This protocol is parameterized identically to $\pi^{\mathsf{Sign}}_{\mathsf{2P\text{-}ECDSA}}$, except that Alice and Bob provide Shamir-shares $p(\mathsf{A}), p(\mathsf{B})$ of sk as input, rather than multiplicative shares.

Key Share Reconstruction:

1) Alice locally calculates the correct Lagrange coefficient $\lambda_{\mathsf{A,B}}$ for Shamir-reconstruction with Bob. Bob likewise calculates $\lambda_{\mathsf{B,A}}$. They then use their respective points $p(\mathsf{A}), p(\mathsf{B})$ on the polynomial p to calculate additive shares of the secret key
$$t^0_{\mathsf{A}} := \lambda_{\mathsf{A,B}} \cdot p(\mathsf{A}) \qquad t^0_{\mathsf{B}} := \lambda_{\mathsf{B,A}} \cdot p(\mathsf{B})$$

Multiplication and Instance Key Exchange:

2) Bob chooses his instance key, $k_{\mathsf{B}} \leftarrow \mathbb{Z}_q$, and Alice chooses her instance key seed, $k'_{\mathsf{A}} \leftarrow \mathbb{Z}_q$. Bob computes $D_{\mathsf{B}} := k_{\mathsf{B}} \cdot G$ and sends D_{B} to Alice.
3) Alice computes
$$\begin{aligned} R' &:= k'_{\mathsf{A}} \cdot D_{\mathsf{B}} \\ k_{\mathsf{A}} &:= H(R') + k'_{\mathsf{A}} \\ R &:= k_{\mathsf{A}} \cdot D_{\mathsf{B}} \end{aligned}$$
4) Alice chooses a pad $\phi \leftarrow \mathbb{Z}_q$, and then Alice and Bob invoke the $\mathcal{F}_{\mathsf{Mul}}$ functionality with inputs $\phi + 1/k_{\mathsf{A}}$ and $1/k_{\mathsf{B}}$ respectively, and receive shares t^1_{A} and t^1_{B} of their padded joint inverse instance key.
5) Alice and Bob invoke the $\mathcal{F}_{\mathsf{Mul}}$ functionality with inputs $t^0_{\mathsf{A}}/k_{\mathsf{A}}$ and $1/k_{\mathsf{B}}$ respectively. They receive shares $t^{2a}_{\mathsf{A}}, t^{2a}_{\mathsf{B}}$ of Alice's secret key share over their joint instance key
$$t^{2a}_{\mathsf{A}} + t^{2a}_{\mathsf{B}} = \frac{t^0_{\mathsf{A}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}}$$
6) Alice and Bob invoke the $\mathcal{F}_{\mathsf{Mul}}$ functionality with inputs $1/k_{\mathsf{A}}$ and $t^0_{\mathsf{B}}/k_{\mathsf{B}}$ respectively. They receive shares $t^{2b}_{\mathsf{A}}, t^{2b}_{\mathsf{B}}$ of Bob's secret key share over their joint instance key
$$t^{2b}_{\mathsf{A}} + t^{2b}_{\mathsf{B}} = \frac{t^0_{\mathsf{B}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}}$$
7) Alice and Bob merge their respective shares
$$t^2_{\mathsf{A}} := t^{2a}_{\mathsf{A}} + t^{2b}_{\mathsf{A}} \qquad t^2_{\mathsf{B}} := t^{2a}_{\mathsf{B}} + t^{2b}_{\mathsf{B}}$$
8) Alice transmits R' to Bob, who computes
$$R := H(R') \cdot D_{\mathsf{B}} + R'$$
For both Alice and Bob let $(r_x, r_y) = R$.
9) Alice submits $(\texttt{prove}, k_{\mathsf{A}}, D_{\mathsf{B}})$ to $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$ and Bob submits $(\texttt{prove}, R, D_{\mathsf{B}})$. Bob receives a bit indicating that the proof was sound. If it was not, he aborts.

Consistency Check, Signature, and Verification:

10) Alice and Bob both compute $m' = H(m)$.
11) Alice computes the first check value Γ^1, encrypts her pad ϕ with Γ^1, and transmits the encryption η^ϕ to Bob.
$$\begin{aligned} \Gamma^1 &:= G + \phi \cdot k_{\mathsf{A}} \cdot G - t^1_{\mathsf{A}} \cdot R \\ \eta^\phi &:= H(\Gamma^1) + \phi \end{aligned}$$
12) Alice computes her share of the signature $\mathsf{sig}_{\mathsf{A}}$ and the second check value Γ^2. She encrypts $\mathsf{sig}_{\mathsf{A}}$ with Γ^2 and then transmits the encryption η^{sig} to Bob
$$\begin{aligned} \mathsf{sig}_{\mathsf{A}} &:= (m' \cdot t^1_{\mathsf{A}}) + (r_x \cdot t^2_{\mathsf{A}}) \\ \Gamma^2 &:= (t^1_{\mathsf{A}} \cdot \mathsf{pk}) - (t^2_{\mathsf{A}} \cdot G) \\ \eta^{\mathsf{sig}} &:= H(\Gamma^2) + \mathsf{sig}_{\mathsf{A}} \end{aligned}$$
13) Bob computes the check values and reconstructs the signature
$$\begin{aligned} \Gamma^1 &:= t^1_{\mathsf{B}} \cdot R \\ \phi &:= \eta^\phi - H(\Gamma^1) \\ \theta &:= t^1_{\mathsf{B}} - \phi/k_{\mathsf{B}} \\ \mathsf{sig}_{\mathsf{B}} &:= (m' \cdot \theta) + (r_x \cdot t^2_{\mathsf{B}}) \\ \Gamma^2 &:= (t^2_{\mathsf{B}} \cdot G) - (\theta \cdot \mathsf{pk}) \\ \mathsf{sig} &:= \mathsf{sig}_{\mathsf{B}} + \eta^{\mathsf{sig}} - H(\Gamma^2) \end{aligned}$$
14) Bob uses the public key pk to verify that $\sigma := (\mathsf{sig}, r_x)$ is a valid signature on message m. If the verification fails, Bob aborts. If it succeeds, he outputs σ.

VI. Multiplication with OT Extensions

Both our 2-of-2 and 2-of-n signing protocols depend upon a functionality that computes an additive sharing of the product of two inputs. We wish the protocol that implements this functionality to be secure against malicious adversaries and practically efficient in the non-amortized setting. Furthermore, if our signing protocols are to be only two messages overall, then our multiplication protocol must comprise a single message from Bob to Alice, followed by a reply, and no further interaction. These requirements preclude generic approaches such SPDZ [51] or MASCOT [52]. Instead, we devise a new variant of the classic Gilboa oblivious multiplication construction [32], which is based upon Oblivious Transfer. Whereas Gilboa's original formulation is only semi-honest secure, our modified technique ensures security against active adversaries, while allowing one party to induce (simulatable) additive errors into the output, which can be detected via the final signature verification in the broader context of our signing scheme. Specifically, our multiplication protocol realizes the following multiplication-with-errors functionality.

Functionality 3. $\mathcal{F}_{\mathsf{Mul}}$**:**

This functionality is parameterized by the group order q. It runs with two parties, Alice and Bob, who may participate in the Init phase once, and the Bob-input and Multiply phases as many times as they wish.

Init: Wait for message $(\texttt{init})$ from Alice and Bob. Store $(\texttt{init-complete})$ in memory and send $(\texttt{init-complete})$ to Bob.

Bob-input: On receiving $(\texttt{input}, \mathsf{id}^{\mathsf{mul}}, \beta)$ from Bob, if $(\texttt{bob-input}, \mathsf{id}^{\mathsf{mul}}, \cdot, \cdot)$ with the same $\mathsf{id}^{\mathsf{mul}}$ does not exist in memory, and if $(\texttt{init-complete})$ does exist in memory, and if $\beta \in \mathbb{Z}_q$, then sample $t_{\mathsf{A}} \leftarrow \mathbb{Z}_q$ uniformly at random, store $(\texttt{bob-input}, \mathsf{id}^{\mathsf{mul}}, \beta, t_{\mathsf{A}})$ in memory, and send $(\texttt{bob-ready}, \mathsf{id}^{\mathsf{mul}}, t_{\mathsf{A}})$ to Alice.

Multiply: On receiving $(\texttt{input}, \mathsf{id}^{\mathsf{mul}}, \alpha, \delta, c)$ from Alice, if there exists a message of the form $(\texttt{bob-input}, \mathsf{id}^{\mathsf{mul}}, \beta, t_{\mathsf{A}})$ in memory with the same $\mathsf{id}^{\mathsf{mul}}$, and if $(\texttt{complete}, \mathsf{id}^{\mathsf{mul}})$ does not exist in memory, and if $\alpha, \delta \in \mathbb{Z}_q$ and $c \in \mathbb{Z}^*$ such that $c \geq 1 \iff \delta \neq 0$, then:

1) Toss c coins, and if any of them output 1, then send $(\texttt{cheat-detected})$ to Bob.
2) Otherwise, calculate $t_{\mathsf{B}} := \alpha \cdot \beta + \delta - t_{\mathsf{A}}$ and send $(\texttt{output}, \mathsf{id}^{\mathsf{mul}}, t_{\mathsf{B}})$ to Bob.
3) Store $(\texttt{complete}, \mathsf{id}^{\mathsf{mul}})$ in memory.

A. Oblivious Transfer

In order to improve the practical efficiency of our algorithm, we base our multiplier upon Correlated Oblivious Transfer Extensions rather than traditional OT. Whereas in plain OT, the sender provides two messages, in Correlated OT, the sender provides one correlation (in our case, an additive correlation), and the messages are generated randomly under this constraint.

What follows is a Correlated OT-extension functionality that allows arbitrarily many Correlated OT instances to be executed in batches of size ℓ. For each batch, the receiver inputs a vector of choice bits $\boldsymbol{\omega} \in \{0,1\}^{\ell}$, following which the sender inputs a vector of correlations $\boldsymbol{\alpha} \in \mathbb{G}_1 \times \mathbb{G}_2 \times \ldots \times \mathbb{G}_{\ell}$ (each element $\boldsymbol{\alpha}_i$ being in some agreed-upon group $\mathbb{G}_i$). The functionality samples ℓ random pads, each pad i being from the corresponding group $\mathbb{G}_i$, and sends them to the sender. To the receiver it sends only the pads if the sender's corresponding choice bits were 0, or the sum of the pads and their corresponding correlations if the sender's corresponding choice bits were 1. Note that this functionality is nearly identical to the one presented by Keller *et al.* [34], but we add flexible correlation lengths, an initialization phase, and the ability to perform extensions (each batch of extensions indexed by a fresh extension index $\mathsf{id}^{\mathsf{ext}}$) only after the initialization has been performed.

Functionality 4. $\mathcal{F}^{\ell}_{\mathsf{COTe}}$**:**

This functionality is parameterized by the group order q and the batch size ℓ. It runs with two parties, a sender S and a receiver R, who may participate in the Init phase once, and the Choice and Transfer phases as many times as they wish.

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

Choice: On receiving $\left(\texttt{choose}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}\right)$ 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 $\boldsymbol{\omega}$ is of the correct form, then send $(\texttt{chosen})$ to the sender and store $\left(\texttt{choice}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}\right)$ in memory.

Transfer: On receiving $\left(\texttt{transfer}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\alpha}\right)$ from the sender, if a message of the form $\left(\texttt{choice}, \mathsf{id}^{\mathsf{ext}}, \boldsymbol{\omega}\right)$ 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 $\boldsymbol{\alpha}$ is of the correct form, then:

1) Sample a vector of random pads $\mathbf{t}_{\mathsf{S}} \leftarrow \mathbb{G}_1 \times \mathbb{G}_2 \times \ldots \times \mathbb{G}_{\ell}$
2) Send $(\texttt{pads}, \mathbf{t}_{\mathsf{S}})$ to the sender.
3) Compute $\{\mathbf{t}_{\mathsf{R}i}\}_{i \in [1,\ell]} := \{\boldsymbol{\omega}_i \cdot \boldsymbol{\alpha}_i - \mathbf{t}_{\mathsf{S}i}\}_{i \in [1,\ell]}$.
4) Send $(\texttt{padded-correlation}, \mathbf{t}_{\mathsf{R}})$ to the receiver.
5) Store $(\texttt{complete}, \mathsf{id}^{\mathsf{ext}})$ in memory.

We instantiate this functionality using the protocol of Keller *et al.* [34]. As with all OT-extension protocols, a base-OT protocol is required. Here we use the Simplest OT protocol of Chou and Orlandi [33], which we modify by adding a new check, to overcome a weakness in the proof of the original formulation. The details of these protocols and of our modifications are given in Appendix A.

B. Single Multiplication

The classic Gilboa OT-multiplication [32] takes an input from Alice and an input from Bob, and returns to them additive secret shares of the product of those two inputs. It works essentially by performing binary multiplication with a single oblivious transfer for each bit in Bob's input. Unfortunately, this protocol is vulnerable to selective failure attacks in the malicious setting. Alice can corrupt one of the two messages during any single transfer, and in doing so learn the value of Bob's input bit for that transfer according to whether or not their outputs are correct. We address this by encoding Bob's input with enough redundancy that learning s (a statistical security parameter) of Bob's choice bits via selective failure does not leak information about the original input value. A bit-by-bit consistency check ensures that the parties abort with high probability if an inconsistent message is selected by Bob, and thus the probability that Alice succeeds in more than s selective failures is exponentially small. A proposition of Impagliazzo and Naor [53] gives us the following encoding scheme: for an input β of length κ, sample $\kappa + 2s$ random bits $\boldsymbol{\gamma} \leftarrow \{0,1\}^{\kappa+2s}$ and take the dot product with some public random vector $\mathbf{g}^{\mathsf{R}} \in \mathbb{Z}_q^{\kappa+2s}$. Use this dot product as a mask for the original input. The encoding function is defined as

Algorithm 4. $\mathsf{Encode}(\mathbf{g}^{\mathsf{R}} \in \mathbb{Z}_q^{\kappa+2s}, \beta \in \mathbb{Z}_q)$**:**

1) Sample $\boldsymbol{\gamma} \leftarrow \{0,1\}^{\kappa+2s}$
2) Output $\mathsf{Bits}\left(\beta - \langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma} \rangle\right) \| \boldsymbol{\gamma}$

In Appendix E, we prove formally that this encoding scheme produces codewords with the property that even if one knows the message encoded, guessing any substring of a codeword is almost as hard as guessing a uniformly sampled string of the same length. We also prove that the following protocol realizes $\mathcal{F}_{\mathsf{Mul}}$.

Protocol 5. Multiplication (π_{Mul})**:**

This protocol is parameterized by the statistical security parameter s, the curve order q, and the symmetric security parameter $\kappa = |q|$. It also makes use of a coefficient vector $\mathbf{g} = \mathbf{g}^{\mathsf{G}} \| \mathbf{g}^{\mathsf{R}}$, where $\mathbf{g}^{\mathsf{G}} \in \mathbb{Z}_q^{\kappa}$ is a *gadget vector* such that $\mathbf{g}^{\mathsf{G}}_i = 2^{i-1}$, and $\mathbf{g}^{\mathsf{R}} \leftarrow \mathbb{Z}_q^{\kappa+2s}$ is a public random vector. It requires access to the Correlated Oblivious Transfer functionality $\mathcal{F}^{\ell}_{\mathsf{COTe}}$. Alice supplies some input integer $\alpha \in \mathbb{Z}_q$, and Bob supplies some input integer $\beta \in \mathbb{Z}_q$. Alice and Bob receive t_{A} and $t_{\mathsf{B}} \in \mathbb{Z}_q$ as output, respectively, such that $t_{\mathsf{A}} + t_{\mathsf{B}} = \alpha \cdot \beta$.

Encoding:

1) Bob encodes his input
$$\boldsymbol{\omega} := \mathsf{Encode}(\mathbf{g}^{\mathsf{R}}, \beta)$$
2) Alice samples $\hat{\alpha} \leftarrow \mathbb{Z}_q$ and sets
$$\boldsymbol{\alpha} := \{\alpha \| \hat{\alpha}\}_{j\in[1,2\kappa+2s]}$$

Multiplication:

3) Alice and Bob access the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ functionality, with batch size $\ell := 2\kappa + 2s$. Alice plays the sender, supplying $\boldsymbol{\alpha}$ as her input, and Bob, the receiver, supplies $\boldsymbol{\omega}$. They receive as outputs, respectively, the arrays
$$\left\{\mathbf{t}_{\mathsf{A}j} \middle\| \hat{\mathbf{t}}_{\mathsf{A}j}\right\}_{j\in[1,2\kappa+2s]} \quad \text{and} \quad \left\{\mathbf{t}_{\mathsf{B}j} \middle\| \hat{\mathbf{t}}_{\mathsf{B}j}\right\}_{j\in[1,2\kappa+2s]}$$
That is, $\mathbf{t}_{\mathsf{A}}$ is a vector wherein each element contains the first half of the corresponding element in Alice's output from $\mathcal{F}^{\ell}_{\mathsf{COTe}}$, and $\hat{\mathbf{t}}_{\mathsf{A}}$ is a vector wherein each element contains the second half. $\mathbf{t}_{\mathsf{B}}$ and $\hat{\mathbf{t}}_{\mathsf{B}}$ play identical roles for Bob.
4) Alice and Bob generate two shared, random values by calling the random oracle. As input they use the shared components of the transcript of the protocol that implements $\mathcal{F}^{\ell}_{\mathsf{COTe}}$, in order the ensure that these values have a temporal dependency on the completion of the previous step. In our proofs, we abstract this step as a coin tossing protocol.
$$(\chi, \hat{\chi}) \leftarrow H^2(\mathsf{transcript})$$
5) Alice computes
$$\mathbf{r} := \left\{\chi \cdot \mathbf{t}_{\mathsf{A}j} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}j}\right\}_{j\in[1,2\kappa+2s]}$$
$$u := \chi \cdot \alpha + \hat{\chi} \cdot \hat{\alpha}$$
and sends $\mathbf{r}$ and u to Bob
6) Bob aborts if
$$\bigvee_{j\in[1,2\kappa+2s]} \left(\chi \cdot \mathbf{t}_{\mathsf{B}j} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{B}j} \neq \boldsymbol{\omega}_j \cdot u - \mathbf{r}_j\right)$$
7) Alice and Bob compute their output shares
$$t_{\mathsf{A}} := \sum_{j\in[1,2\kappa+2s]} \mathbf{g}_j \cdot \mathbf{t}_{\mathsf{A}j} \qquad t_{\mathsf{B}} := \sum_{j\in[1,2\kappa+2s]} \mathbf{g}_j \cdot \mathbf{t}_{\mathsf{B}j}$$

C. Coalesced Multiplication

The multiplication protocol described in the foregoing section supports the multiplication of only a single integer α by a single integer β, and in our two-party and 2-of-n signing protocols ($\pi^{\mathsf{Sign}}_{\mathsf{2P\text{-}ECDSA}}$ and $\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}$ respectively) we invoke the multiplication protocol two or three times. An optimization allows these multiple invocations to be combined at reduced cost, albeit by breaking some of our previous abstractions.

Consider first the case of two-party signing, wherein two multiplications must be performed. Each multiplication individually encodes its input, enlarging it by $\kappa + 2s$ bits, and then individually calls upon the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ Correlated OT-extension functionality with batch size $\ell = 2\kappa + 2s$. The protocol that realizes this functionality incurs some overhead, proportionate to a security parameter κ^{OT}, and two multiplications performed in the naïve way incur this cost twice. However, we observe that two multiplication protocol instances can share a single invocation of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ simply by doubling the batch size, thereby reducing the extension cost by an amount proportionate to κ^{OT}. Furthermore, we observe that when the inputs are combined into a single extension instance, we can also combine the encodings of the inputs, reducing the overhead due to encoding from $2\kappa + 4s$ additional OT instances to $2\kappa + 2s$. In a future version of this paper, we show that this coalesced encoding maintains security.

Algorithm 5. $\mathsf{Encode2}(\mathbf{g}^{\mathsf{R}} \in \mathbb{Z}_q^{\kappa+2s}, \beta^1 \in \mathbb{Z}_q, \beta^2 \in \mathbb{Z}_q)$**:**

1) Sample $\boldsymbol{\gamma}^1 \leftarrow \{0,1\}^{\kappa}, \boldsymbol{\gamma}^2 \leftarrow \{0,1\}^{\kappa}, \boldsymbol{\gamma}^3 \leftarrow \{0,1\}^{2s}$
2) Output
$$\mathsf{Bits}(\beta^1 - \langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma}^1 \| \boldsymbol{\gamma}^3 \rangle) \| \boldsymbol{\gamma}^1$$
$$\| \ \mathsf{Bits}(\beta^2 - \langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma}^2 \| \boldsymbol{\gamma}^3 \rangle) \| \boldsymbol{\gamma}^2 \| \boldsymbol{\gamma}^3$$

Further consider the case of 2-of-n signing, in which three multiplications are used to compute the products
$$\alpha^1 \cdot \beta^1 \qquad \alpha^{2a} \cdot \beta^2 \qquad \alpha^{2b} \cdot \beta^1$$
Notice that in the first and third multiplications, Bob's inputs are identical, while in the second it differs. Consequently, we can perform the third multiplication by extending the appropriate part Alice's input, while keeping Bob's input the same.

Protocol 6. Coalesced Triple Multiplication (π_{Mul3})**:**

This protocol is parameterized identically to π_{Mul}, except that Alice supplies three inputs, $\alpha^1, \alpha^{2a}, \alpha^{2b}$ and receives three outputs, $t^1_{\mathsf{A}}, t^{2a}_{\mathsf{A}}, t^{2b}_{\mathsf{A}}$. Bob supplies only two inputs, β^1, β^2, and likewise receives $t^1_{\mathsf{B}}, t^{2a}_{\mathsf{B}}, t^{2b}_{\mathsf{B}}$.

Encoding:

1) Bob encodes his input
$$\boldsymbol{\omega} := \mathsf{Encode2}(\mathbf{g}^{\mathsf{R}}, \beta^1, \beta^2)$$
2) Alice samples $\hat{\alpha}^1, \hat{\alpha}^{2a}, \hat{\alpha}^{2b} \leftarrow \mathbb{Z}_q$ and sets
$$\boldsymbol{\alpha} := \left\{\alpha^1 \middle\| \hat{\alpha}^1 \middle\| \alpha^{2a} \middle\| \hat{\alpha}^{2a}\right\}_{j\in[1,2\kappa]}$$
$$\| \left\{\alpha^{2b} \middle\| \hat{\alpha}^{2b}\right\}_{j\in[1,2\kappa]}$$
$$\| \left\{\alpha^1 \middle\| \hat{\alpha}^1 \middle\| \alpha^{2a} \middle\| \hat{\alpha}^{2a} \middle\| \alpha^{2b} \middle\| \hat{\alpha}^{2b}\right\}_{j\in[1,2s]}$$

Multiplication:

3) Alice and Bob access the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ functionality, with batch size $\ell := 4\kappa + 2s$. Alice plays the sender, supplying $\boldsymbol{\alpha}$

as her input, and Bob, the receiver, supplies $\boldsymbol{\omega}$. Alice receives as output the array

$$\left\{\mathbf{t}^{1}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{1}_{\mathsf{A}j}\middle\|\mathbf{t}^{2\mathrm{a}}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{2\mathrm{a}}_{\mathsf{A}j}\right\}_{j\in[1,2\kappa]}$$
$$\|\left\{\mathbf{t}^{2\mathrm{b}}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{2\mathrm{b}}_{\mathsf{A}j}\right\}_{j\in[1,2\kappa]}$$
$$\|\left\{\mathbf{t}^{1}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{1}_{\mathsf{A}j}\middle\|\mathbf{t}^{2\mathrm{a}}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{2\mathrm{a}}_{\mathsf{A}j}\middle\|\mathbf{t}^{2\mathrm{b}}_{\mathsf{A}j}\middle\|\hat{\mathbf{t}}^{2\mathrm{b}}_{\mathsf{A}j}\right\}_{j\in[2\kappa,2\kappa+2s]}$$

and Bob receives a corresponding output array.

The remainder of the protocol is identical to π_{Mul}, except that the linear check process is repeated for each of the tuples

$$\left(\mathbf{t}^{1}_{\mathsf{A}},\mathbf{t}^{1}_{\mathsf{B}},\hat{\mathbf{t}}^{1}_{\mathsf{A}},\hat{\mathbf{t}}^{1}_{\mathsf{B}}\right)\quad\left(\mathbf{t}^{2\mathrm{a}}_{\mathsf{A}},\mathbf{t}^{2\mathrm{a}}_{\mathsf{B}},\hat{\mathbf{t}}^{2\mathrm{a}}_{\mathsf{A}},\hat{\mathbf{t}}^{2\mathrm{a}}_{\mathsf{B}}\right)\quad\left(\mathbf{t}^{2\mathrm{b}}_{\mathsf{A}},\mathbf{t}^{2\mathrm{b}}_{\mathsf{B}},\hat{\mathbf{t}}^{2\mathrm{b}}_{\mathsf{A}},\hat{\mathbf{t}}^{2\mathrm{b}}_{\mathsf{B}}\right)$$

To compute three products in the naïve way, $\kappa\cdot(3\kappa^{\mathsf{OT}}+24\kappa+24s+9)$ bits must be transferred, with a proportionate amount of computation being performed. Using our optimized, coalesced multiplication, only $\kappa\cdot(\kappa^{\mathsf{OT}}+22\kappa+20s+5)$ bits must be transferred (again, with a proportionate amount of computation). Concretely, if we use $\kappa=256$, $s=80$, and $\kappa^{\mathsf{OT}}=128+s$ (this being the overhead induced by the OT-extension protocol; our choice follows KOS [34]), then the total communication is reduced from to 271.8 to 232.7 KiB.

VII. Cost Analysis

When all of the optimizations have been applied and all functionalities and sub-protocols have been collapsed, we find that our protocols have communication and computation costs as reported in Table I. Though we account completely for communications, we count only elliptic curve point multiplications and calls to the hash function H toward computation cost. We assume that both commitments and the PRG are implemented via the random oracle H, and that proofs-of-knowledge-of-discrete-logarithm are implemented via Schnorr protocols with the Fiat-Shamir heuristic.

The 2-of-n setup protocol is somewhat more complex than Table I indicates. Over its course, each of the n parties commits to and then sends a single proof-of-knowledge-of-discrete-logarithm to all other parties in broadcast and then verifies the $n-1$ proofs that it receives. The parties then compute and send Lagrange coefficients to one another, which requires $O(n^2)$ (parallel) communication in total, and this pattern repeats for verification. Finally, each party evaluates a single KOS Setup instance with every other party, for $(n^2-n)/2$ instances in total. The entire protocol requires four broadcast rounds, plus the messages required by the KOS Setup instances.

For ease of comparison, concrete communication costs for our signing protocol along with the signing protocols of Gennaro *et al.* [4], Boneh *et al.* [5], and Lindell [3] are listed in Table II. The former pair of schemes are related: Boneh *et al.* reduce the number of messages in Gennaro *et al.*'s signing protocol from six to four, with the goal of reducing the communication cost. Apart from requiring only two messages, our signing protocol requires roughly one seventh of the communication incurred by either.

Lindell's signing scheme requires four messages and excels in terms of communication cost, only transferring a commitment, two curve points, two zero-knowledge proofs, and one Paillier ciphertext. However, the Paillier homomorphic operations it requires are quite expensive. Lindell's scheme requires one encryption, one homomorphic scalar multiplication, and one homomorphic addition with a Paillier modulus $N>2q^4+q^3$; concretely, a standard 2048-bit modulus is sufficient for a 256-bit curve. Gennaro *et al.* and Boneh *et al.*'s schemes both require one to three encryptions and three to five homomorphic additions and scalar multiplications per party, with $N>q^8$, which likewise implies that for 256-bit curves, a 2048-bit modulus is sufficient. In addition, Lindell's protocol requires 12 Elliptic Curve multiplications, while the protocols of the other two require roughly 100. These Paillier and curve operations dominate the computation cost of the protocols.

VIII. Implementation

We created a proof-of-concept implementation of our 2-of-2 and 2-of-n setup and signing protocols in the Rust language. As a prerequisite, we also created an elliptic curve library in Rust. We use SHA-256 to instantiate the random oracle H, per the ECDSA specification, and in addition we use it to instantiate the PRG. As a result, our protocol makes no concrete cryptographic assumptions other than those already required by ECDSA itself. The SHA-256 implementation used in signing is capable of parallelizing vectors of hash operations, and the 2-of-n setup protocol is capable of parallelizing OT-extension initializations, but otherwise the code is strictly single-threaded. This approach has likely resulted in reduced performance relative to an optimized C implementation, but we believe that the safety afforded by Rust makes the trade worthwhile.

We benchmarked our implementation on a pair of Amazon `C5.2xlarge` instances from Amazon's Virginia datacenter, both running Ubuntu 16.04 with Linux kernel 4.4.0, and we compiled our code using Rust 1.27 with the default level of optimization. The bandwidth between our instances was measured to be be 5GBits/Second, and the round-trip latency to be 0.2 ms. Our signatures were calculated over the secp256k1 curve, as standardized by NIST [6]. Thus $\kappa=256$, and we chose $s=80$ and $\kappa^{\mathsf{OT}}=128+s$, following the analysis of KOS [34]. We performed both strictly single-threaded benchmarks, and benchmarks allowing parallel hashing with three threads per party, collecting 10,000 samples for setup and 100,000 for signing. Note that signatures were not batched, and thus each sample was impacted individually by the full latency of the network. The average wall-clock times for both signing protocols and the 2-of-2 setup protocol are reported in Table III, along with results from previous works for comparison. These results are taken directly from their respective sources, and were not produced in our benchmarking environment. Nevertheless, we believe them to be comparable, due to the fact that they were collected using a similar type of hardware and in similar network conditions.

We benchmarked our 2-of-n setup algorithm using set of 20 Amazon `C5.2xlarge` instances from the Virginia datacenter,

	Rounds	Communication (Bits)	EC Multiplications		Hash Function Invocations	
			Alice	Bob	Alice	Bob
2-of-2 Setup	5	$\kappa \cdot (5\kappa + 11) + 6$	$3\kappa + 6$	$2\kappa + 6$	$6\kappa + 4$	$6\kappa + 4$
2-of-2 Signing	2	$\kappa \cdot (\kappa^{\mathsf{OT}} + 16\kappa + 14s + 10) + 3$	7	9	$2\kappa^{\mathsf{OT}} + 24\kappa + 20s + 9$	$3\kappa^{\mathsf{OT}} + 20\kappa + 14s + 9$
2-of-n Signing	2	$\kappa \cdot (\kappa^{\mathsf{OT}} + 22\kappa + 20s + 11) + 3$	7	9	$2\kappa^{\mathsf{OT}} + 32\kappa + 28s + 9$	$3\kappa^{\mathsf{OT}} + 24\kappa + 18s + 9$
			Max	Min	Max	Min
2-of-n Setup	5	$(n^2 - n) \cdot (2.5\kappa^2 + 8\kappa + 4)$	$n\kappa - \kappa + 4$	$n + 3$	$5n\kappa - 5\kappa + 1$	$4n\kappa - 4\kappa + 1$

TABLE I: **Communication and Computation Cost Equations For Our Protocol**. We assume that the hash function H is used to implement the PRG. Note that communication costs are totals for all parties over all rounds, whereas computation costs are given per party. In the 2-of-n protocol the computation cost depends upon the identity of the party; consequently we give the minimum and maximum.

	$\kappa = 256$	$\kappa = 384$	$\kappa = 521$
Lindell [3]	769 B	897 B	1043 B
This Work (2-of-2)	169.8 KiB	350.7 KiB	615.3 KiB
Gennaro *et al.* [4]	~1808 KiB	~4054 KiB	~7454 KiB
Boneh *et al.* [5]	~1680 KiB	~3768 KiB	~6924 KiB
This Work (2-of-n)	232.8 KiB	481.3 KiB	844.7 KiB

TABLE II: **Concrete Signing Communication Costs**. Assuming 2-of-n signing for Gennaro *et al.* and Boneh *et al.*, and 2-of-2 signing for the protocol of Lindell. For our protocols, we use $s = 80$ and $\kappa^{\mathsf{OT}} = 128 + s$.

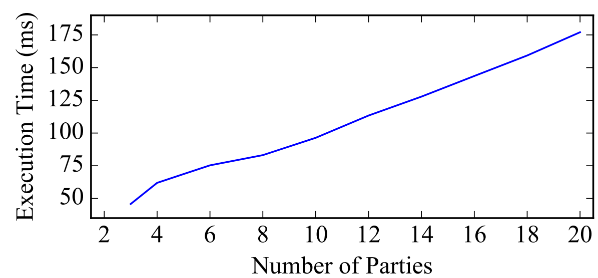

Fig. 2: **Wall Clock Times for 2-of-n Setup over LAN**. Note that all 20 parties reside on individual machines in the same datacenter, and latency is on the order of a few tenths of a millisecond.

	This Work	(3 threads)	[3]	
2-of-2 Setup	43.41	–	2435	
2-of-2 Signing	3.26	3.12	36.8	
	This Work	(3 threads)	[4]	[5]
2-of-n Signing	3.77	3.55	~650	~350

TABLE III: **Wall-clock Times in Milliseconds over LAN**, as compared to the prior approaches of Lindell [3], Gennaro *et al.* [4], and Boneh *et al.* [5]. Note that hardware and networking environments are not necessarily equivalent, but all benchmarks were performed with a single thread except where specified.

Setup			Signing	
2-of-2	2-of-4 (US)	2-of-10 (World)	2-of-2	2-of-n
354.36	376.86	1228.46	81.34	81.83

TABLE IV: **Wall-clock Times in Milliseconds over WAN**. All benchmarks were performed between one party in the eastern US and one in Paris, except the 2-of-4 setup benchmark, which was performed among four parties in four different US states, and the 2-of-10 setup benchmark, which was performed among ten parties in America, Europe, Asia, and Australia.

configured as before with one instance per party. For initializing OT-extensions, each machine was allowed to use as many threads as there were parties, but the code was otherwise single-threaded. We collected 1000 samples for groups of parties ranging in size from 3 to 20, and we report the results in Figure 2.

Transoceanic Benchmarks: We repeated our 2-of-2 setup, 2-of-2 signing, and 2-of-n signing benchmarks with one of the machines relocated to Amazon's Paris datacenter, collecting 1,000 samples for setup and 10,000 for signing, and in the latter case allowing three threads for hashing. In this configuration, the bandwidth between our instances was measured to be 155 Mbps and the round-trip latency to be 78.2 ms. In addition, we performed a 2-of-4 setup benchmark among four instances in Amazon's four US datacenters (Virginia, Ohio, California, and Oregon), and we performed a 2-of-10 setup benchmark among ten instances in ten geographically distributed datacenters (Virginia, Ohio, California, Oregon, Mumbai, Sydney, Canada, Ireland, London, and Paris). The round-trip latency between the US datacenters was between 11.2 ms and 79.9 ms and the bandwidth between 152 Mbps and 1.10 Gbps, while round-trip latency between the most distant pair of datacenters, Mumbai and Ireland, was 282 ms, and the bandwidth was 39 Mbps. Results are reported in Table IV. We note that in contrast to our single-datacenter benchmarks, our transoceanic benchmarks are dominated by latency costs. We expect that our protocol's low round count constitutes a greater advantage in this setting than does its computational efficiency.

A. Comparison to Prior Work

We compare our implementation to those of Lindell [3], Gennaro *et al.* [4], and Boneh *et al.* [5] (who also provide an optimized version of Gennaro *et al.*'s scheme, against which we make our comparison). Though Boneh *et al.* and Gennaro *et al.* support thresholds larger than two, we consider only their performance in the 2-of-n case. Neither Gennaro *et al.* nor Boneh *et al.* include network costs in the timings they provide, nor do they provide timings for the setup protocol that their schemes share. However, Lindell observes that Gennaro *et al.*'s scheme involves a distributed Paillier key generation protocol that requires roughly 15 minutes to run in the semi-honest setting. Unfortunately, this means we have no reliable point of comparison for our 2-of-n setup protocol.

Lindell benchmarks his scheme using a single core on each of two Microsoft Azure `Standard_DS3_v2` instances in the same datacenter, which can expect bandwidth of roughly 3 Gbps. Lindell's performance figures do include network costs. In spite of the fact that Lindell's protocol requires vastly less communication, as reported in Section VII, we nonetheless find that, not accounting for differences in benchmarking environment, our implementation outperforms his for signing by a factor of roughly 11 (when only a single thread is allowed), and for setup by a factor of roughly 56.

Given that each 2-of-2 signature requires 169.8 KiB of data to be transferred under our scheme, but only 769 Bytes under Lindell's, there must be an environment in which his scheme outperforms ours. Specifically Lindell has an advantage when the protocol is bandwidth constrained but not computationally constrained. Such a scenario is likely when a large number of signatures must calculated in a batched fashion (mitigating the effects of latency) by powerful machines with a comparatively weak network connection.

Finally, we note that an implementation of the ordinary (local) ECDSA signing algorithm in Rust using our own elliptic curve library requires an average of 173 microseconds to calculate a signature on our benchmark machines – a factor of roughly 18 faster than our 2-of-2 signing protocol.

IX. Acknowledgments

We thank Megan Chen and Emily Wang for their contributions to this project during the summer of 2017. The authors of this work are supported by NSF grants TWC-1646671 and TWC-1664445. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) Jetstream cluster [54] through allocation TG-CCR170010, which is supported by NSF grant number ACI-1548562.

X. Code Availability

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

References

[1] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *IEEE S&P*, 2018.

[2] Y. Desmedt, "Society and group oriented cryptography: A new concept," in *CRYPTO*, 1987.

[3] Y. Lindell, *Fast Secure Two-Party ECDSA Signing*, 2017.

[4] R. Gennaro, S. Goldfeder, and A. Narayanan, *Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security*, 2016.

[5] D. Boneh, R. Gennaro, and S. Goldfeder, "Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security," http://www.cs.haifa.ac.il/~orrd/LC17/paper72.pdf, 2017.

[6] 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.

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

[8] D. R. L. Brown, "Sec 2: Recommended elliptic curve domain parameters," 2010. [Online]. Available: http://www.secg.org/sec2-v2.pdf

[9] D. Kravitz, "Digital signature algorithm," jul 1993, uS Patent 5,231,668.

[10] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic curve digital signature algorithm (dsa) for dnssec," https://tools.ietf.org/html/rfc4492, 2006.

[11] P. Hoffman and W. Wijngaards, "Elliptic curve digital signature algorithm (dsa) for dnssec," https://tools.ietf.org/html/rfc6605, 2012.

[12] Bitcoin Wiki, "Transaction," https://en.bitcoin.it/wiki/Transaction, 2017, accessed Oct 22, 2017.

[13] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2017. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[14] Y. G. Desmedt and Y. Frankel, "Threshold cryptosystems," in *CRYPTO*, 1989.

[15] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO*, 1984.

[16] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, Nov. 1979.

[17] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *EUROCRYPT*, 1991.

[18] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures (extended abstract)," in *CRYPTO*, 1991.

[19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, Feb. 1978.

[20] Y. Desmedt and Y. Frankel, "Parallel reliable threshold multisignature," 1992.

[21] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust and efficient sharing of rsa functions," in *CRYPTO*, 1996.

[22] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely," in *STOC*, 1994.

[23] V. Shoup, "Practical threshold signatures," in *EUROCRYPT*, 2000.

[24] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, 1989.

[25] D. R. Stinson and R. Strobl, "Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates," in *ACISP*, 2001.

[26] S. K. Langford, "Threshold dss signatures without a trusted party," in *CRYPTO*, 1995.

[27] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold dss signatures," in *EUROCRYPT*, 1996.

[28] P. MacKenzie and M. K. Reiter, *Two-Party Generation of DSA Signatures*, 2001.

[29] Bitcoin Wiki, "Multisignature," https://en.bitcoin.it/wiki/Multisignature, 2017, accessed Oct 22, 2017.

[30] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[31] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, Sep. 1976.

[32] N. Gilboa, "Two party rsa key generation," in *CRYPTO*, 1999.

[33] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, 2015.

[34] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *CRYPTO*, 2015.

[35] E. Hauck and J. Loss, "Efficient and universally composable protocols for oblivious transfer from the cdh assumption," Cryptology ePrint Archive, Report 2017/1011, 2017, http://eprint.iacr.org/2017/1011.

[36] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2015, ch. Digital Signature Schemes, pp. 443–486.

[37] D. R. L. Brown, "Generic groups, collision resistance, and ecdsa," Cryptology ePrint Archive, Report 2002/026, 2002, http://eprint.iacr.org/2002/026.
[38] S. Vaudenay, "The security of dsa and ecdsa," in *PKC*, 2003.
[39] N. Koblitz and A. Menezes, "Another look at generic groups," Cryptology ePrint Archive, Report 2006/230, 2006, https://eprint.iacr.org/2006/230.
[40] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, Jun. 1985.
[41] M. O. Rabin, "How to exchange secrets with oblivious transfer," Cryptology ePrint Archive, Report 2005/187, 1981, http://eprint.iacr.org/2005/187, Harvard University Technical Report 81.
[42] S. Wiesner, "Conjugate coding," *SIGACT News*, 1983.
[43] M. Naor and B. Pinkas, "Computationally secure oblivious transfer," *J. Cryptol.*, vol. 18, no. 1, Jan. 2005.
[44] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," in *STOC*, 1996.
[45] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, and A. Sahai, "Efficient non-interactive secure computation," in *EUROCRYPT*, 2011.
[46] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky, "Non-interactive secure multiparty computation," in *CRYPTO*, 2014.
[47] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *EUROCRYPT*, 1997.
[48] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.
[49] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *CRYPTO*, 2005.
[50] E. Waring, *Philosophy Transactions*, no. 69, pp. 59–67, 1779.
[51] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.
[52] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," in *CCS*, 2016.
[53] R. Impagliazzo and M. Naor, "Efficient cryptographic schemes provably as secure as subset sum," *J. Cryptol.*, vol. 9, no. 4, Sep 1996.
[54] C. Stewart, T. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, M. Vaughn, and N. Gaffney, "Jetstream: a self-provisioned, scalable science and engineering cloud environment," in *XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, 2015. [Online]. Available: http://dx.doi.org/10.1145/2792745.2792774
[55] M. Byali, A. Patra, D. Ravi, and P. Sarkar, "Efficient, round-optimal, universally-composable oblivious transfer and commitment scheme with adaptive security," Cryptology ePrint Archive, Report 2017/1165, 2017, https://eprint.iacr.org/2017/1165.
[56] P. S. L. M. Barreto, B. David, R. Dowsley, K. Morozov, and A. C. A. Nascimento, "A framework for efficient adaptively secure composable oblivious transfer in the rom," Cryptology ePrint Archive, Report 2017/993, 2017, https://eprint.iacr.org/2017/993.
[57] M. Fischlin, "A note on security proofs in the generic model," in *ASIACRYPT*, 2000.
[58] J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart, "Flaws in applying proof methodologies to signature schemes," in *Advances in Cryptology — CRYPTO 2002*, M. Yung, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 93–110.
[59] A. W. Dent, "Adapting the weaknesses of the random oracle model to the generic group model," in *Advances in Cryptology — ASIACRYPT 2002*, Y. Zheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 100–109.
[60] D. Boneh and X. Boyen, "Short signatures without random oracles," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–73.
[61] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 41–55.
[62] E. Bresson, O. Chevassut, and D. Pointcheval, *New Security Results on Encrypted Key Exchange*, 2004.
[63] R. Canetti, A. Jain, and A. Scafuro, "Practical uc security with a global random oracle," in *CCS*, 2014.
[64] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, 2001.
[65] R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo-random generation from one-way functions," in *STOC*, 1989.

APPENDIX A
OBLIVIOUS TRANSFER

We augment Simplest OT [33] with a verification procedure and refer to the new primitive as Verified Simplest OT (VSOT). VSOT is used as the basis for an instantiation of the KOS [34] OT-extension protocol, which is used in turn to build the OT-multiplication primitive required by our main signing protocol.

If we did not desire simulation-based malicious security, then it may have been sufficient to use the Simplest OT scheme without modification. In composing the protocol to build a larger simulation-sound malicious protocol however, there is a complication. The security proof relies upon the fact that the protocol's hash queries are modeled as calls to a random oracle, and uses those queries to extract the receiver's inputs. However, the queries need not occur before the receiver has sent its last message, and so there is no guarantee that a malicious receiver will actually query the oracle. When Simplest OT is composed, it may be the case that the receiver's inputs are required for simulation before they are required by the receiver itself, in which case the protocol will be unsimulatable. This flaw has recently been noticed by a number of authors, and we refer the reader to other works [35], [55], [56] for more detailed discussions. Barreto *et al.* [56] propose to solve the problem by adding a public-key verification process in the Random Oracle Model. Rather than using expensive public-key operations, however, we specify that the receiver must prove knowledge of its output using only symmetric-key operations, ensuring that it does in fact hold that output, and therefore that its input is extractable. As a consequence, our protocol is able to realize only an OT functionality ($\mathcal{F}_{\mathsf{SF\text{-}OT}}$) that allows for selective failure by the sender, but we show that this is sufficient for our purposes.

A. Verified Simplest OT

We begin by describing the VSOT protocol. Because Alice and Bob participate in this protocol with their roles reversed, relative to the usual arrangement, we refer to the participants simply as the sender and receiver in this section. The protocol comprises four phases. In the first, the sender generates a private/public key pair, and sends the public key to the receiver. In the second phase, the receiver encodes its choice bit and the sender generates two random pads based upon the encoded choice bit in such a way that the receiver can only recover one. The third phase is a verification, which is necessary to ensure that the protocol is simulatable. Finally, the pads are used by the sender to mask its messages for transmission to the receiver in the fourth phase. This protocol realizes the $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ functionality, which is given in Appendix B.

Protocol 7. Verified Simplest OT (π_{VSOT}):

This protocol is parameterized by the Elliptic curve $(\mathbb{G}, G, q)$, and symmetric security parameter $\kappa = |q|$. It relies upon the $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ functionality, and makes use of a hash function H. It takes as input a choice bit $\omega \in \{0, 1\}$ from the receiver, and two messages $\alpha^0, \alpha^1 \in \mathbb{Z}_q$ from the sender. It outputs one message $\alpha^\omega \in \mathbb{Z}_q$ to the receiver, and nothing to the sender.

Public Key:

1) The sender samples $b \leftarrow \mathbb{Z}_q$, computes $B := b \cdot G$, and transmits B to the receiver.
2) The sender submits $(\texttt{prove}, b, G)$ to the $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ functionality. The receiver submits $(\texttt{prove}, B, G)$, and receives a bit indicating whether the proof was sound. If it was not, the receiver aborts.

Pad Transfer:

3) The receiver samples $a \leftarrow \mathbb{Z}_q$, and then computes its encoded choice bit A and the pad ρ^ω

$$A := a \cdot G + \omega \cdot B$$
$$\rho^\omega := H(a \cdot B)$$

and sends A to the sender.

4) The sender computes two pads

$$\rho^0 := H(b \cdot A)$$
$$\rho^1 := H(b \cdot (A - B))$$

Verification:

5) The sender computes a challenge

$$\xi := H(H(\rho^0)) \oplus H(H(\rho^1))$$

and sends the challenge ξ to the receiver.

6) The receiver computes a response

$$\rho' := H(H(\rho^\omega)) \oplus (\omega \cdot \xi)$$

and sends ρ' to the sender.

7) The sender aborts if $\rho' \neq H(H(\rho^0))$. Otherwise, it opens its challenge by sending $H(\rho^0)$ and $H(\rho^1)$ to the receiver.
8) The receiver aborts if the value of $H(\rho^\omega)$ it received from the sender does not match the one it calculated itself, or if

$$\xi \neq H(H(\rho^0)) \oplus H(H(\rho^1))$$

Message Transfer:

9) The sender pads its two messages α^0, α^1, and transmits the padded messages $\tilde{\alpha}^0, \tilde{\alpha}^1$ to the receiver

$$\tilde{\alpha}^0 := \alpha^0 + \rho^0$$
$$\tilde{\alpha}^1 := \alpha^1 + \rho^1$$

10) The receiver removes the pad from its chosen message

$$\alpha^\omega = \tilde{\alpha}^\omega - \rho^\omega$$

For simplicity, we describe VSOT as requiring one complete protocol evaluation per OT instance. However, if (public) nonces are used in each of the hash invocations, then the Public Key phase can be run once and the resulting (single) public key B can be reused in as many Transfer and Verification phases as required without sacrificing security. Further note that if the messages transmitted by the sender are specified to be uniform, then the sender can actually omit the Message Transfer phase entirely and treat the pads ρ^0, ρ^1 as messages, receiving them as output instead of supplying them as input. Likewise, the receiver treats its one pad ρ^ω as its output. This effectively transforms VSOT into a Random OT protocol. We make use of both of these optimizations in our implementation.

B. Correlated OT-extension with KOS

Our multiplication protocol requires the use of a large number of OT instances where the correlation between messages is specified, but the messages must otherwise be random. Therefore, rather than using VSOT directly, we layer a Correlated OT-extension (COTe) protocol atop it. This is essentially an instantiation of the KOS protocol; thus we include a protocol description here for completeness, but refer the reader to Keller *et al.* [34] for a more thorough discussion. Being a Correlated OT protocol, it allows the sender to define a correlation between the two messages, but does not allow the sender to determine the messages specifically. As with all OT-extension systems, it is divided into a setup protocol, which uses some base OT system to generate correlated secrets between the two parties, and an extension protocol, which uses these correlated secrets to efficiently perform additional OTs. These protocols realize the Correlated Oblivious Transfer functionality $\mathcal{F}_{\mathsf{COTe}}^{\ell}$, which is given in Section VI.

Protocol 8. KOS Setup $\left(\pi_{\mathsf{KOS}}^{\mathsf{Setup}}\right)$:

This protocol is parameterized by the curve order q and the symmetric security parameter $\kappa = |q|$. It depends upon the OT Functionality $\mathcal{F}_{\mathsf{SF\text{-}OT}}$, and takes no input from either party. Alice receives as output a private OTe correlation $\boldsymbol{\nabla} \in \{0,1\}^\kappa$ and a vectors of seeds $\mathbf{s}^{\boldsymbol{\nabla}} \in \mathbb{Z}_q^\kappa$, and Bob receives two vectors of seeds $\mathbf{s}^0$ and $\mathbf{s}^1 \in \mathbb{Z}_q^\kappa$.

Setup:

1) Alice samples a correlation vector, $\boldsymbol{\nabla} \leftarrow \{0,1\}^\kappa$.
2) For each bit $\boldsymbol{\nabla}_i$ of the correlation vector, Alice and Bob access the $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ functionality, with Alice acting as the receiver and using $\boldsymbol{\nabla}_i$ for her choice bit and Bob acting as the sender. Bob samples two random seed elements $\mathbf{s}_i^0 \leftarrow \mathbb{Z}_q$ and $\mathbf{s}_i^1 \leftarrow \mathbb{Z}_q$ and Alice receives as output a single seed element $\mathbf{s}_i^{\boldsymbol{\nabla}_i}$.
3) Alice and Bob collate their individual seed elements into vectors, $\mathbf{s}^{\boldsymbol{\nabla}}$ and $\mathbf{s}^0, \mathbf{s}^1$ respectively, and take these vectors as output.

Protocol 9. KOS Extension $\left(\pi_{\mathsf{KOS}}^{\mathsf{Extend}}\right)$:

This protocol is parameterized by the OT batch size ℓ, the OT security parameter κ^{OT}, the curve order q, and the symmetric security parameter $\kappa = |q|$. For notational convenience, let $\ell' = \ell + \kappa^{\mathsf{OT}}$. It makes use of the pseudo-random generator $\mathsf{Prg}_{\mathbb{Z}} : \mathbb{Z}_q^\kappa \mapsto \mathbb{Z}_{2^{\ell'}}$, which expands its argument and then outputs the chunk of ℓ' bits indexed by the value given as a subscript, and it makes use use of the hash function H. The protocol also uses a fresh, public OT-extension index, $\mathsf{id}^{\mathsf{ext}}$. Alice supplies a vector of input integers, $\boldsymbol{\alpha} \in \mathbb{G}_1 \times \mathbb{G}_2 \times \ldots \times \mathbb{G}_\ell$, along with her private OTe correlation $\boldsymbol{\nabla} \in \{0,1\}^\kappa$

and seed $\mathbf{s}^{\boldsymbol{\nabla}} \in \mathbb{Z}_q^\kappa$, which she received during the KOS setup protocol. Bob supplies a vector of choice bits $\boldsymbol{\omega} \in \{0,1\}^\ell$ along with his seeds $\mathbf{s}^0$ and $\mathbf{s}^1 \in \mathbb{Z}_q^\kappa$ from the OT setup. Alice and Bob receive $\mathbf{t}_{\mathsf{A}}$ and $\mathbf{t}_{\mathsf{B}} \in \mathbb{G}_1 \times \mathbb{G}_2 \times \ldots \times \mathbb{G}_\ell$ as output.

Extension:

1) Bob chooses $\boldsymbol{\gamma}^{\mathsf{ext}} \leftarrow \{0,1\}^{\kappa^{\mathsf{OT}}}$ and collates $\mathbf{w} := \boldsymbol{\omega} \| \boldsymbol{\gamma}^{\mathsf{ext}}$. We use w to indicate $\mathbf{w}$ interpreted as a single value in $\mathbb{Z}_{2^{\ell'}}$. That is, $\mathsf{Bits}(w) = \mathbf{w}$.
2) Bob computes two vectors of PRG expansions of his OT-extension seeds
$$\mathbf{v}^0 := \left\{\mathsf{Prg}_{\mathsf{id}^{\mathsf{ext}}}(\mathbf{s}_i^0)\right\}_{i\in[1,\kappa]}$$
$$\mathbf{v}^1 := \left\{\mathsf{Prg}_{\mathsf{id}^{\mathsf{ext}}}(\mathbf{s}_i^1)\right\}_{i\in[1,\kappa]}$$
and Alice computes a vector of expansions of her correlated seed
$$\mathbf{v}^{\boldsymbol{\nabla}} := \left\{\mathsf{Prg}_{\mathsf{id}^{\mathsf{ext}}}(\mathbf{s}_i^{\boldsymbol{\nabla}_i})\right\}_{i\in[1,\kappa]}$$
3) Bob collates the vector $\boldsymbol{\psi} \in \mathbb{Z}_q^{\ell'}$, which is the transpose of $\mathbf{v}^0$. That is, the first element of $\boldsymbol{\psi}$ is the concatenation of the first bits of all of the elements of $\mathbf{v}^0$, and so on. More formally if we define a matrix
$$\mathbf{V} \in \{0,1\}^{\kappa\times\ell'}$$
then the relationship is given by
$$\mathbf{V}^i = \mathsf{Bits}(\mathbf{v}_i^0) \quad \forall i \in [1,\kappa]$$
$$\mathbf{V}_j = \mathsf{Bits}(\boldsymbol{\psi}_j) \quad \forall j \in [1,\ell']$$
4) Bob computes the matrix
$$\mathbf{u} := \left\{\mathbf{v}_i^0 \oplus \mathbf{v}_i^1 \oplus w\right\}_{i\in[1,\kappa]}$$
and then he computes a matrix of pseudo-random elements from $\mathbb{Z}_q$
$$\boldsymbol{\chi} := \{H(j\|\mathbf{u})\}_{j\in[1,\ell']}$$
which he uses to create a linear sampling of $\mathbf{w}$ and $\boldsymbol{\psi}$
$$w' := \bigoplus_{j\in[1,\ell']} \mathbf{w}_j \cdot \boldsymbol{\chi}_j$$
$$v' := \bigoplus_{j\in[1,\ell']} \boldsymbol{\psi}_j \wedge \boldsymbol{\chi}_j$$
Finally, he sends w', v', and $\mathbf{u}$ to Alice.
5) Alice computes the vector
$$\mathbf{z} := \left\{\mathbf{v}_i^{\boldsymbol{\nabla}_i} \oplus (\boldsymbol{\nabla}_i \cdot \mathbf{u}_i)\right\}_{i\in[1,\kappa]}$$
and collates the vector $\boldsymbol{\zeta}$, which is the transpose of $\mathbf{z}$ in exactly the way that $\boldsymbol{\psi}$ is the transpose $\mathbf{v}^0$. She also calculates $\boldsymbol{\chi}$ in the same manner as Bob
$$\boldsymbol{\chi} := \{H(j\|\mathbf{u})\}_{j\in[1,\ell']}$$
Finally, she computes
$$z' := \bigoplus_{j\in[1,\ell']} \boldsymbol{\zeta}_j \wedge \boldsymbol{\chi}_j$$
and if $z' \neq v' \oplus (\nabla \wedge w')$, where ∇ is $\boldsymbol{\nabla}$ reinterpreted as an element in $\mathbb{Z}_{2^\kappa}$, then Alice aborts.

Transfer:

6) Alice computes
$$\mathbf{t}_{\mathsf{A}} := \left\{H^{|\boldsymbol{\alpha}_j|/\kappa}(j\|\boldsymbol{\zeta}_j)\right\}_{j\in[1,\ell]}$$
$$\boldsymbol{\tau} := \left\{H^{|\boldsymbol{\alpha}_j|/\kappa}(j\|(\boldsymbol{\zeta}_j \oplus \nabla)) - \mathbf{t}_{\mathsf{A}j} + \boldsymbol{\alpha}_j\right\}_{j\in[1,\ell]}$$
and sends $\boldsymbol{\tau}$ to Bob
7) Bob computes
$$\mathbf{t}_{\mathsf{B}} := \left\{\begin{cases} -H^{|\boldsymbol{\tau}_j|/\kappa}(j\|\boldsymbol{\psi}_j) & \text{if } \mathbf{w}_j = 0 \\ \boldsymbol{\tau}_j - H^{|\boldsymbol{\tau}_j|/\kappa}(j\|\boldsymbol{\psi}_j) & \text{if } \mathbf{w}_j = 1 \end{cases}\right\}_{j\in[1,\ell]}$$

Appendix B
Additional Functionalities

In this section, we present the additional functionalities on which our protocols rely. As before, we omit notation for bookkeeping elements that we do not explicitly use such as session IDs and party specifiers, which work in the ordinary way; we also assume that if messages are received out of order for a particular session, the functionality aborts. We begin with a Selective-failure OT functionality, which differs from the traditional OT functionality in that it allows the sender to guess the receiver's choice bit. If the sender's guess is incorrect, the functionality alerts both parties, and if the sender's guess is correct, then the sender is notified while the receiver is not.

Functionality 5. $\mathcal{F}_{\mathsf{SF\text{-}OT}}$**:**

This functionality is parameterized by the group order q and runs with two parties, a sender and a receiver.

Choose: On receiving $(\texttt{choose}, \omega)$ from the receiver, store $(\texttt{choice}, \omega)$ if no such message exists in memory and send $(\texttt{chosen})$ to the sender.

Guess: On receiving $(\texttt{guess}, \hat{\omega})$ from the sender, if $\hat{\omega} \in \{0,1,\perp\}$ and if $(\texttt{choice}, \omega)$ exists in memory, and if $(\texttt{guess}, \cdot)$ does not exist in memory, then store $(\texttt{guess}, \hat{\omega})$ in memory and do the following:

1) If $\hat{\omega} = \perp$, send $(\texttt{no-cheat})$ to the receiver.
2) If $\hat{\omega} = \omega$, send $(\texttt{cheat-undetected})$ to the sender and $(\texttt{no-cheat})$ to the receiver.
3) Otherwise, send $(\texttt{cheat-detected})$ to both the sender and receiver.

Transfer: On receiving $(\texttt{transfer}, \alpha^0, \alpha^1)$ from the sender, if $\alpha^0 \in \mathbb{Z}_q$ and $\alpha^1 \in \mathbb{Z}_q$, and if $(\texttt{complete})$ does not exist in memory, and if there exist in memory messages $(\texttt{choice}, \omega)$ and $(\texttt{guess}, \hat{\omega})$ such that $\hat{\omega} = \perp$ or $\hat{\omega} = \omega$, then send $(\texttt{message}, \alpha^\omega)$ to the receiver and store $(\texttt{complete})$ in memory.

Finally, we give functionalities for zero-knowledge proofs-of-knowledge-of-discrete-logarithm. The first corresponds to an ordinary proof, whereas the second allows the prover to commit to a proof that will later be revealed. Note that these are standard constructions, except that they operate with groups of parties, and all parties aside from the prover receive verification.

Functionality 6. $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$**:**

The functionality is parameterized by the group $\mathbb{G}$ of order q generated by G, and runs with a group of parties $\mathcal{P}$ such that $|\mathcal{P}| = n$.

Proof: On receiving $(\texttt{prove}, x, B_i)$ from $\mathcal{P}_i$ where $x \in \mathbb{Z}_q$ and $B_i \in \mathbb{G}$, store this message and the index i. On receiving $\left(\texttt{prove}, X, B_j\right)$ from $\mathcal{P}_j$ where $X, B_j \in \mathbb{G}$, if $X = x \cdot B_i = x \cdot B_j$, then send $(\texttt{accept}, i)$ to $\mathcal{P}_j$. Otherwise, send $(\texttt{fail}, i)$ to $\mathcal{P}_j$. Note that multiple parties $\mathcal{P}_j$ may participate.

Functionality 7. $\mathcal{F}_{\mathsf{Com\text{-}ZK}}^{R_{\mathsf{DL}}}$**:**

The functionality is parameterized by the group $\mathbb{G}$ of order q generated by G, and runs with a group of parties $\mathcal{P}$ such that $|\mathcal{P}| = n$.

Commit Proof: On receiving $(\texttt{com-proof}, x, B_i)$ from $\mathcal{P}_i$, where $x \in \mathbb{Z}_q$ and $B_i \in \mathbb{G}$, store $(\texttt{com-proof}, x, B_i)$ and send $(\texttt{committed}, i)$ to all parties.

Decommit Proof: On receiving $(\texttt{decom-proof})$ from $\mathcal{P}_i$, store this message in memory. On receiving $\left(\texttt{prove}, X, B_j\right)$ from $\mathcal{P}_j$ where $X, B_j \in \mathbb{G}$, if $(\texttt{com-proof}, x, B_i)$ and $(\texttt{decom-proof})$ exist in memory, then:

1) If $X = x \cdot B_i = x \cdot B_j$, send $(\texttt{accept}, i)$ to $\mathcal{P}_j$.
2) Otherwise send $(\texttt{fail}, i)$ to $\mathcal{P}_j$.

Note that multiple parties $\mathcal{P}_j$ may participate.

APPENDIX C
EQUIVALENCE OF FUNCTIONALITIES

We argue that $\mathcal{F}_{\mathsf{SampledECDSA}}$ does not grant any additional power to Alice by showing that an adversary who is able to forge a signature by accessing $\mathcal{F}_{\mathsf{SampledECDSA}}$ can be used to forge an ECDSA signature in the standard Existential Unforgeability experiment that defines security for signature schemes (see Katz and Lindell [36] for a complete description of the experiment). We are only concerned with arguing that an ideal adversary interacting with $\mathcal{F}_{\mathsf{SampledECDSA}}$ as *Alice* is unable to forge a signature because Bob's view in his ideal interaction with $\mathcal{F}_{\mathsf{SampledECDSA}}$ is identical to his view when interacting with $\mathcal{F}_{\mathsf{ECDSA}}$.

Our reduction is in the Generic Group Model, which was introduced by Shoup [47]. While there are well-known criticisms of this model [57]–[59], it has also shown itself to be useful in proving the security of well-known constructions such as Short Signatures [60] and Short Group Signatures [61]. Furthermore, this is the model in which ECDSA itself is proven secure [37].

In this model an adversary can perform group operations only by querying a Group Oracle $\mathcal{G}(\cdot)$. More specifically, queries of the following types are answered by the Oracle:

1) (Group Elements) When the Oracle receives an integer $x \in \mathbb{Z}_q$, it replies with an encoding of the group element corresponding to this integer. Returned encodings are random, but the Oracle is required to be consistent when the same integer is queried repeatedly. This corresponds to the scalar multiplication operation with the generator in an ECDSA group: $Y \coloneqq x \cdot G$.
2) (Group Law) When the Oracle receives a tuple of the form $(r, s, \mathcal{G}(x), \mathcal{G}(y))$, it replies with a random encoding of the group element given by $\mathcal{G}(r \cdot x + s \cdot y)$. As before, outputs must be consistent. This corresponds to a fused multiply-add operation in an ECDSA group: $Z \coloneqq (r \cdot X + s \cdot Y)$, where $X = x \cdot G$ and $Y = y \cdot G$.

As usual in this model, the reduction itself will control the Group Oracle, and in particular it has the ability to program the Oracle to respond to specific queries with specific outputs.

$\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ is used to denote an Oracle version of the $\mathcal{F}_{\mathsf{SampledECDSA}}$ functionality accessible only as Alice. In addition to the previously defined $\mathcal{F}_{\mathsf{SampledECDSA}}$ behavior, this Oracle returns the signature $\sigma_{\mathsf{id}^{\mathsf{sig}}}$ to Alice upon receiving $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, \cdot, \cdot)$. This models the realistic scenario wherein Alice obtains the output signatures, which we wish to capture in our reduction, even though the functionality does not output the signature to her on its own.

Claim C.1. *If there exists a probabilistic polynomial time algorithm* A *in the Generic Group Model with access to the* $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ *oracle, such that*

$$\Pr\left[\begin{array}{r}\mathsf{Verify}_{\mathsf{pk}}\left(m,\sigma\right)=1\wedge m\notin\mathbf{Q}:\\ (m,\sigma)\leftarrow\mathsf{A}^{\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}}(\mathsf{pk})\end{array}\right]\geq p(\kappa)$$

where $\mathbf{Q}$ *is the set of messages for which* A *sends queries of the form* $(new, \cdot, m, \cdot)$ *to the* $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ *Oracle, and where the probability is taken over the randomness of the* $\mathcal{F}_{\mathsf{SampledECDSA}}$ *functionality, then there exists an adversary* $\mathcal{A}$ *such that*

$$\Pr_{\mathsf{pk},\mathsf{sk}}\left[\begin{array}{r}\mathsf{Verify}_{\mathsf{pk}}\left(m,\sigma\right)=1\wedge m\notin\mathbf{Q}:\\ (m,\sigma)\leftarrow\mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})\end{array}\right]\geq p(\kappa)-\frac{\mathrm{poly}(\kappa)}{2^{-\kappa}}$$

where $\mathbf{Q}$ *is the set of messages for which* $\mathcal{A}$ *queries the signing oracle* $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$.

Proof sketch. Our reduction is structured in an intuitive way. For readability we refer to A as Alice in its interactions with $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$, and we note that $\mathcal{A}$ can only interact with Alice on behalf of the $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ Oracle. First, $\mathcal{A}$ forces Alice to accept the same public key that it received externally in the forgery game, and then, for each query Alice makes to her $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ oracle, $\mathcal{A}$ can request a corresponding signature from the $\mathsf{Sign}_{\mathsf{sk}}$ oracle under the same secret key. The nonce R^{sig} in the signature received from $\mathsf{Sign}_{\mathsf{sk}}$ will not match the nonce R that Alice instructs the $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ oracle to use. However, $\mathcal{A}$ can take advantage of the fact that $\mathcal{F}_{\mathsf{SampledECDSA}}{}^{\mathsf{A}}$ is allowed to offset the nonce R by a random value k^{Δ} of its choosing. $\mathcal{A}$ sets k^{Δ} so that $k^{\Delta} \cdot G$ is exactly the difference between R and R^{sig}. Computing k^{Δ} directly would require $\mathcal{A}$ to know the discrete log of the R^{sig} value it was given by the $\mathsf{Sign}_{\mathsf{sk}}$ oracle; instead, $\mathcal{A}$ uses its ability to program the Group Oracle to ensure that $\mathcal{G}(k^{\Delta})$ is the difference between R and the corresponding R^{sig}. We describe $\mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}$ formally below.

Algorithm 6. $\mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$**:**

1) Answer any query $\mathcal{G}(x)$ as $x \cdot G$, and any query $\mathcal{G}(r, s, \mathcal{G}(x), \mathcal{G}(y))$ as $r \cdot \mathcal{G}(x) + s \cdot \mathcal{G}(y)$ unless otherwise explicitly programmed at those points.
2) Send $(\texttt{public-key}, \mathsf{pk})$ to Alice.
3) When a message of the form $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{B})$ is received from Alice, sample $k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}} \leftarrow \mathbb{Z}_q$, calculate $D_{\mathsf{B}} := k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}} \cdot G$, store $(\texttt{sig-message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}})$ in memory, and reply to Alice with
$$(\texttt{nonce-shard}, \mathsf{id}^{\mathsf{sig}}, D_{\mathsf{B}})$$
4) When a message of the form $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_{i,\mathsf{id}^{\mathsf{sig}}})$ is received from Alice, if $(\texttt{sig-message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}})$ exists in memory:
   a) Query the Signing Oracle with the message m to obtain a signature
   $$\left(\mathsf{sig}_{\mathsf{id}^{\mathsf{sig}},i}, R^{\mathsf{sig}}_{\mathsf{id}^{\mathsf{sig}},i}\right) = \sigma_{\mathsf{id}^{\mathsf{sig}},i} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$$
   Note that the oracle will only return the x-coordinate of $R^{\mathsf{sig}}_{\mathsf{id}^{\mathsf{sig}},i}$, but recovering the point itself is easy. Store $(\texttt{sig-signature}, \mathsf{id}^{\mathsf{sig}}, \sigma_{\mathsf{id}^{\mathsf{sig}},i})$ in memory.
   b) Sample $k^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i} \leftarrow \mathbb{Z}_q$, then compute
   $$K^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i} := R^{\mathsf{sig}}_{i,\mathsf{id}^{\mathsf{sig}}} - R_{i,\mathsf{id}^{\mathsf{sig}}}$$
   and program the Group Oracle such that
   $$\mathcal{G}\left(k^{\Delta}_{\mathsf{id}^{\mathsf{sig}}}\right) = K^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i}$$
   c) Compute
   $$k^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i,\mathsf{A}} = (1/k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}}) \cdot k^{\Delta}_{\mathsf{id}^{\mathsf{sig}}}$$
   and program the Group Oracle such that
   $$\mathcal{G}(k^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i,\mathsf{A}}) = (1/k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}}) \cdot K^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i}$$
   d) Send $(\texttt{offset}, \mathsf{id}^{\mathsf{sig}}, k^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i,\mathsf{A}})$ to Alice.
5) When a message of the form $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, i, k_{\mathsf{A}})$ is received from Alice, if $(\texttt{sig-signature}, \mathsf{id}^{\mathsf{sig}}, \sigma_{\mathsf{id}^{\mathsf{sig}},i})$ and $(\texttt{sig-message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}})$ exist in memory, and $k_{\mathsf{A}} \cdot k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}} \cdot G = R^{\mathsf{sig}}_{i\mathsf{id}^{\mathsf{sig}}}$, but $(\texttt{sig-complete}, \mathsf{id}^{\mathsf{sig}})$ does not exist in memory, respond with $\sigma_{\mathsf{id}^{\mathsf{sig}},i}$ and store $(\texttt{sig-complete}, \mathsf{id}^{\mathsf{sig}})$ in memory.
6) Once Alice outputs a forged signature sig^*, output this signature.

Notice that this reduction fails if Alice queries $\mathcal{G}$ on an index $k^{\Delta}_{\mathsf{id}^{\mathsf{sig}},i,\mathsf{A}}$ for any $\mathsf{id}^{\mathsf{sig}}$ and any i before $\mathcal{A}$ programs it, or if she queries it on an index $k_{\mathsf{B}}^{\mathsf{id}^{\mathsf{sig}}}$ for any $\mathsf{id}^{\mathsf{sig}}$ at any time. By a standard argument, this event occurs with probability $\mathrm{poly}(\kappa)/2^{\kappa}$. If these queries are not made, the reduction is perfect and the claim follows. □

Appendix D
Proof of Security for Oblivious Transfer

In this section, we argue for the UC-security of the π_{VSOT} protocol discussed in Appendix A, and later discuss the security of the KOS OT-extension protocol when considered in combination with the $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ functionality that π_{VSOT} realizes.

Theorem D.1. *Assuming that the Computational Diffie-Hellman problem is hard in $\mathbb{G}$, the protocol π_{VSOT} UC-realizes the $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ functionality in the $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$-hybrid model in the presence of a statically corrupted malicious party, where H is modeled as a non-programmable random oracle.*

Proof sketch. We now provide simulators for π_{VSOT}, along with an argument for their indistinguishably from the protocol. First, we will consider the case of security against malicious sender; later, we argue security against a malicious receiver.

Simulator 1. VSOT against Sender $(\mathcal{S}^{\mathsf{S}}_{\mathsf{VSOT}})$**:**

This simulator interposes between a malicious sender and the corresponding ideal functionality $\mathcal{F}_{\mathsf{SF\text{-}OT}}$. It is parameterized by the symmetric security parameter κ. It outputs the sender's messages α^0 and α^1. It makes use of the random oracle H, and plays the role of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ in its interactions with the sender.

Public Key:

1) Receive $(\texttt{prove}, b, B)$ from the sender on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ and forward it to $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$. On receiving $(\texttt{accept}, i, B)$, forward it to the sender. If $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ responds with $(\texttt{fail}, i, B)$, then abort.

Pad Transfer:

2) Upon receiving $(\texttt{chosen})$ from $\mathcal{F}_{\mathsf{SF\text{-}OT}}$, sample $A \leftarrow \mathbb{G}$. Send A to the sender and calculate ρ^0 and ρ^1
$$\rho^0 := H(b \cdot A)$$
$$\rho^1 := H(b \cdot (A - B))$$

Verification:

3) Compute sender's expected challenge
$$\xi^{\mathsf{exp}} := H(H(\rho^0)) \oplus H(H(\rho^1))$$
Upon receiving the sender's actual challenge ξ, if $\xi = \xi^{\mathsf{exp}}$, then set $\hat{\omega} := \perp$ and send $(\texttt{guess}, \hat{\omega})$ to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ and $\rho' := H(H(\rho^0))$ to the sender. Otherwise, let $\mathbf{Q}$ denote the set of all queries that the sender has made to the random oracle thus far. If there exists a query $\mathbf{Q}_i \in \mathbf{Q}$ such that $H(\mathbf{Q}_i) = \xi \oplus H(H(\rho^1))$, then set $\hat{\omega} := 1$. Otherwise, set $\hat{\omega} := 0$. Send $(\texttt{guess}, \hat{\omega})$ to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ and receive either $(\texttt{cheat-detected})$ or $(\texttt{cheat-undetected})$, indicating whether the sender succeeded in guessing the receiver's input. If the sender has succeeded and $(\texttt{cheat-undetected})$ is received,

send $\rho' \coloneqq H(H(\rho^{\hat{\omega}}))$ to the sender. Otherwise, send $\rho' \coloneqq H(H(\rho^{\bar{\hat{\omega}}}))$ to the sender and halt.

Transfer:

4) Upon receiving $\tilde{\alpha}^0$ and $\tilde{\alpha}^1$, compute the sender's inputs

$$\alpha^0 \coloneqq \tilde{\alpha}^0 - \rho^0$$
$$\alpha^1 \coloneqq \tilde{\alpha}^1 - \rho^1$$

Finally, send $\left(\texttt{transfer}, \alpha^0, \alpha^1\right)$ to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$.

The first message received by the sender comprises $A = a \cdot G + \omega \cdot G$ in the real world, and $A = a \cdot G$ in the simulation. Because a is picked uniformly in both views, the two are distributed identically. Given A and B, an honest sender computes ρ^0 and ρ^1 as $\rho^0 \coloneqq b \cdot A$ and $\rho^1 \coloneqq b \cdot A \cdot B^{-1}$. Having received b on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$, the simulator can also compute these values, and thus can check whether the value of ξ that it receives is correct. We now consider the sender's view when ξ is not correct.

During the verification phase, the sender is required to open the two values ($H(\rho^0)$ and $H(\rho^1)$) that produce ξ. Only one of these values, $H(\rho^\omega)$, will be known to the receiver. Note that this is the the sender's opportunity to induce a selective failure: the sender can guess which value the receiver has, and substitute a random value for the opposite one when calculating (and later opening) ξ. If the receiver does not abort, the sender's guess was correct. However, a corrupt sender can guess a well-formed triple of values $H(\rho^0), H(\rho^1)$, and ξ without calling $H(H(\rho^0))$ and $H(H(\rho^1))$ with a probability of $2^{-\kappa}$, and if the triple is not well-formed, then the receiver will always abort in the real world. This forces the sender to query the random oracle, and its queries can be used by the simulator to determine the sender's guess $\hat{\omega}$. This is forwarded to the functionality, which informs the simulator whether the guess is correct. From this point, the simulator replies with exactly the same values and aborts under exactly the same conditions as the protocol in the real world. Therefore, the view of a malicious sender when executing π_{VSOT} in the real world is distinguishable from the view of a malicious sender when interacting with $\mathcal{S}_{\mathsf{VSOT}}^{\mathsf{S}}$ with probability no greater than $2^{-\kappa}$.

Now we consider security against a malicious receiver. For this section of the proof sketch, we need an additional lemma

Lemma D.2 ([33], [62]). *Let q be the order of a group $\mathbb{G}$ generated by G, whose elements are represented in $\kappa = |q|$ bits. If there exists a PPT algorithm $\mathcal{A}$ such that:*

$$\Pr[\mathcal{A}(1^\kappa, x \cdot G) = x \cdot x \cdot G : x \leftarrow \mathbb{Z}_q] = \varepsilon$$

where the probability is taken over the choice of x, then there exists an algorithm $\mathcal{A}'$ which solves the Computational Diffie-Hellman problem in $\mathbb{G}$ with advantage ε^2.

Simulator 2. VSOT against Receiver $\left(\mathcal{S}_{\mathsf{VSOT}}^{\mathsf{R}}\right)$:

This simulator interposes between a malicious receiver and the corresponding ideal functionality $\mathcal{F}_{\mathsf{SF\text{-}OT}}$, and is parameterized by the symmetric security parameter κ. It outputs the receiver's choice bit ω and the corresponding chosen message α^ω. It makes use of the random oracle H and the functionality $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$.

Public Key:

1) Sample $b \leftarrow \mathbb{Z}_q$ and compute $B \coloneqq b \cdot G$. Send $(\texttt{accept}, i, B)$ to the receiver on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$.

Pad Transfer:

2) Receive A from the receiver and compute ρ^0, ρ^1 as an honest sender would.
3) Observe receiver's random oracle queries. If receiver ever queries $b \cdot A$, then set $\omega \coloneqq 0$. If receiver ever queries $b \cdot (A - B)$, then set $\omega \coloneqq 1$. Once ω is set, send $(\texttt{choose}, \omega)$ to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ and receive $(\texttt{no-cheat})$.

Verification:

4) Run the verification phase as an honest sender would.

Transfer:

5) Upon receiving $(\texttt{message}, \alpha^\omega)$, compute the two padded messages $\tilde{\alpha}^\omega$, $\tilde{\alpha}^{\bar{\omega}}$, and send them to the receiver.

$$\tilde{\alpha}^\omega \coloneqq \alpha^\omega + \rho^\omega$$
$$\tilde{\alpha}^{\bar{\omega}} \leftarrow \mathbb{Z}_q$$

Fixing $B = b \cdot G$ (which is chosen by the simulator on behalf of the sender) and A (which is chosen and transmitted by the receiver) also fixes the two pads, $\rho^0 = H(b \cdot A)$ and $\rho^1 = H(b \cdot (A - B))$. The receiver cannot derive ρ^ω without querying the random oracle, except with probability $2^{-\kappa}$. The simulator observes the receiver's queries and checks for equality with either $b \cdot A$ or $b \cdot (A - B)$ in order to determine the receiver's choice bit. If the receiver manages to query both values, then the simulator cannot determine its choice bit. We argue that if the receiver can do so with non-negligible probability, then it can be used to compute $x \cdot x \cdot G$ given $X = x \cdot G$ with non-negligible probability in the following way.

Given a uniformly drawn challenge $X = x \cdot G$, we can generate the receiver's view with $B = X$ (which has the correct distribution). We have assumed that the receiver manages to query both $b \cdot A = x \cdot A$ and $b \cdot (A - B) = x \cdot (A - X) = x \cdot A - x \cdot X$, and we have assumed that the receiver can make at most polynomially-many queries to the random oracle. Thus, we can choose any two of the receiver's random oracle queries, and with probability $1/\operatorname{poly}(\kappa)$ their difference will be equal to $x \cdot x \cdot G$. By Lemma D.2, successfully computing $x \cdot x \cdot G$ given $X = x \cdot G$ with non-negligible probability implies breaking the Computational Diffie-Hellman assumption.

The rest of the simulator behaves exactly as the protocol does, and the values it produces are identically distributed to their real-world counterparts. Thus, the view produced by the simulator is computationally indistinguishable from the view of the receiver in a real-world protocol if the CDH problem is hard in $\mathbb{G}$. □

Remark. It is easy to extend the above protocol to implement a Selective-Failure OT functionality that supports the sending of arbitrary sender-chosen messages (instead of the functionality choosing the messages) via the standard reduction from Random OT to OT; that is, by using the sender's output from the Random OT to encrypt the messages that it wants to send. Interestingly, this comes at the cost of no additional rounds. We observe that the resulting Selective Failure OT protocol can be proven secure in the Global Random Oracle Model of Canetti *et al.* [63], in spite of the fact that if the functionality $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ is instantiated with the Fischlin Transform [49], as is required to achieve non-interactivity and UC-security simultaneously, then it typically requires the random oracle to be programmed when simulating against a corrupt verifier. In our case, the value B (for which the prover is required to prove knowledge of discrete logarithm in Step 2 of π_{VSOT}) is chosen by the simulator itself when simulating against the verifier. Consequently, $\mathcal{S}_{\mathsf{VSOT}}^{\mathsf{R}}$ can simply run the honest prover's code to generate a proof, without programming the random oracle at any point. In the context of the security reduction to the Computational Diffie-Hellman Assumption for the receiver that we have given above, it is necessary to simulate $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{DL}}}$ in the traditional way, and therefore to program the random oracle. This is not a problem, as it is legal to program the (global) random oracle in a reduction.

Finally, we note that our implementation of π_{VSOT} reuses the first message of the sender across multiple *parallel* instances, which, as discussed by Chou and Orlandi [33] realizes the same functionality adjusted for multiple messages.

Lemma D.3. *The OT Extension protocol of Keller et al. [34] ($\pi_{\mathsf{KOS}}^{\mathsf{Setup}}$ and $\pi_{\mathsf{KOS}}^{\mathsf{Extend}}$) UC-realizes the $\mathcal{F}_{\mathsf{COTe}}^{\ell}$ functionality in the $\mathcal{F}_{\mathsf{SF\text{-}OT}}$-hybrid model where H is modeled as a non-programmable random oracle.*

Proof sketch. As these protocols are nearly exactly the same as in Keller *et al.* [34], their analysis applies unmodified. We observe that weakening the OT functionality used by Keller *et al.* to our Selective Failure OT functionality does not allow the malicious receiver in their protocol any additional advantage. For each $(\mathtt{guess}, \hat{\omega})$ message that a malicious receiver sends to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ in our protocol, a malicious receiver in the protocol of Keller *et al.* would send (m_0, m_1) with $m_{\hat{\omega}} = \bot$. We recall that the receiver is allowed to learn c bits of the sender's private correlation $\boldsymbol{\nabla}$, however at the risk of alerting the sender of a cheat with probability $1-2^{-c}$ (as the sender's input to $\mathcal{F}_{\mathsf{SF\text{-}OT}}$ is uniform). As the correlation $\boldsymbol{\nabla}$ is broken in the Extend of the protocol phase by being passed through a random oracle, the receiver's overall success probability in making a correct random oracle query (corresponding to the key she is not supposed to derive) is $\mathrm{poly}(\kappa)/2^{\kappa}$. □

Remark. We observe that the OT extension protocol of Keller *et al.* can be simulated without programming the random oracle. Therefore, when instantiated with our π_{VSOT} protocol, it realizes the $\mathcal{F}_{\mathsf{COTe}}^{\ell}$ functionality in the Global Random Oracle Model under only the Computational Diffie-Hellman assumption. To our knowledge, this is the first realization of OT extension in this model and under this assumption.

APPENDIX E
PROOF OF SECURITY FOR MULTIPLICATION

In this section, we prove that the multiplication protocol π_{Mul} realizes the $\mathcal{F}_{\mathsf{Mul}}$ functionality in the Global Random Oracle Model [63] under the Universal Composability (UC) paradigm. For a definition of UC-security, we refer the reader to the seminal work of Canetti [64]. In Appendix E-A we prove Lemma E.2, which states that the view of Alice in π_{Mul} is simulatable, and in Appendix E-B, Lemma E.4 makes a similar claim with regard to the view of Bob. By the conjunction of these two lemmas, we claim Theorem E.1.

Theorem E.1. *The protocol π_{Mul} UC-realizes the functionality $\mathcal{F}_{\mathsf{Mul}}$ in the $\mathcal{F}_{\mathsf{COTe}}^{\ell}$-hybrid Random Oracle Model, in the presence of a malicious adversary statically corrupting either party.*

A. Simulating Against Alice

Simulator 3. Multiplication against Alice $\left(\mathcal{S}_{\mathsf{Mul}}^{\mathsf{A}}\right)$:

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

Init:

1) Receive message $(\mathtt{init})$ from Alice on behalf of $\mathcal{F}_{\mathsf{COTe}}^{\ell}$ and send $(\mathtt{init})$ to $\mathcal{F}_{\mathsf{Mul}}$.

Multiplication:

2) Upon receiving $(\mathtt{bob\text{-}ready}, \mathsf{id}^{\mathsf{mul}}, t_{\mathsf{A}})$ from $\mathcal{F}_{\mathsf{Mul}}$, send $(\mathtt{chosen}, \mathsf{id}^{\mathsf{mul}})$ to Alice on behalf of $\mathcal{F}_{\mathsf{COTe}}^{\ell}$.
3) Receive $(\mathtt{transfer}, \mathsf{id}^{\mathsf{mul}}, \{\boldsymbol{\alpha}_i \| \hat{\boldsymbol{\alpha}}_i\}_{i\in[1,\ell]})$ from Alice on behalf of $\mathcal{F}_{\mathsf{COTe}}^{\ell}$, and receive her messages u and $\mathbf{r}$. Engage in the coin tossing protocol (corresponding to Step 4 of π_{Mul}) with Alice to derive the values χ and $\hat{\chi}$.
4) For each $i \in [1, \ell]$, compute
$$\boldsymbol{\Delta}_i := \chi \cdot \boldsymbol{\alpha}_i + \hat{\chi} \cdot \hat{\boldsymbol{\alpha}}_i - u$$
Next, compile a list $\mathbf{I}$ of the locations where Alice has cheated. If, for any $i \in [1, \ell]$,
$$\boldsymbol{\Delta}_i \neq 0 \vee \mathbf{r}_i \neq \chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i}$$
then append i to $\mathbf{I}$ and compute
$$\tilde{\boldsymbol{\omega}}_i := \frac{\mathbf{r}_i - (\chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i})}{\boldsymbol{\Delta}_i}$$
If there exists any index i such that $\tilde{\boldsymbol{\omega}}_i$ is defined but not in $\{0, 1\}$, then abort by setting $\delta := 0$ and $c := \kappa$ and skipping to Step 6. Otherwise, set $c := |\mathbf{I}|$ so that the $\mathcal{F}_{\mathsf{Mul}}$ functionality will abort with probability $1 - 2^{-|\mathbf{I}|}$.

5) Choose any index $i \in [1, \ell]$ such that $\chi \cdot \boldsymbol{\alpha}_i + \hat{\chi} \cdot \hat{\boldsymbol{\alpha}}_i = u$ and let $\alpha := \boldsymbol{\alpha}_i$. If no such index exists, then abort by setting $\delta := 0$, $c := \kappa$ and skipping to Step 6. Otherwise, compute the additive offset

$$\delta := \sum_{i \in \mathbf{I}} \tilde{\boldsymbol{\omega}}_i \cdot \mathbf{g}_i \cdot (\boldsymbol{\alpha}_i - \alpha)$$

6) Send $(\texttt{input}, \mathsf{id}^{\mathsf{mul}}, \alpha, \delta, c)$ to $\mathcal{F}_{\mathsf{Mul}}$ and halt

Lemma E.2. *The view produced by $\mathcal{S}^{\mathsf{A}}_{\mathsf{Mul}}$ and the view of a malicious Alice in a real execution of the protocol π_{Mul} are indistinguishable to any probabilistic polynomial time adversary in the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$-hybrid Random Oracle Model, except with negligible probability.*

Proof. Our proof of Lemma E.2 will proceed via a sequence of hybrid experiments. The information in Alice's view is characterized by the values $\mathbf{t}_{\mathsf{A}}$ and $\hat{\mathbf{t}}_{\mathsf{A}}$ that she receives as output from $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ upon sending it $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\alpha}}$. The joint distribution of outputs over which distinguishability is considered for the purpose of this proof includes t_{A} and t_{B}, as well as a bit indicating whether Bob has been induced to abort.

For our first hybrid, we will need a lemma concerning the distribution of encodings of Bob's input. Recall that Bob encodes his input using Algorithm 4:

Algorithm 4. $\mathsf{Encode}(\mathbf{g}^{\mathsf{R}} \in \mathbb{Z}_q^{\kappa+2s}, \beta \in \mathbb{Z}_q)$**:**

1) Sample $\boldsymbol{\gamma} \leftarrow \{0,1\}^{\kappa+2s}$
2) Output $\mathsf{Bits}\left(\beta - \langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma} \rangle\right) \| \boldsymbol{\gamma}$

We wish to show that it is hard to guess an encoding $\boldsymbol{\omega} \leftarrow \mathsf{Encode}(\mathbf{g}^{\mathsf{R}}, \beta)$, even given β. A guess comprises a vector $\hat{\boldsymbol{\omega}} \in \{0, 1, \perp\}^{\ell}$, where $\ell = 2\kappa + 2s$ is the size of an encoding, and the $\perp$ symbol indicates that no guess is made for a particular bit. A guess *matches* an encoding if and only if $\hat{\boldsymbol{\omega}}_i = \boldsymbol{\omega}_i \vee \hat{\boldsymbol{\omega}}_i = \perp$ for all indices $i \in [1, \ell]$.

Lemma E.3. *Given $\mathbf{g}^{\mathsf{R}} \leftarrow \mathbb{Z}_q^{\kappa+2s}$, it holds with overwhelmingly high probability that for all values $\beta \in \mathbb{Z}_q$ and $\hat{\boldsymbol{\omega}} \in \{0, 1, \perp\}^{\ell}$ such that $g = |\{\hat{\boldsymbol{\omega}}_i \in \hat{\boldsymbol{\omega}} : \hat{\boldsymbol{\omega}}_i \neq \perp\}|$,*

$$\Pr\left[\mathsf{Encode}(\mathbf{g}^{\mathsf{R}}, \beta) \text{ matches } \hat{\boldsymbol{\omega}}\right] \leq 2^{-g} + 2^{-s}$$

where the probability is taken over the randomness of the encoding.

Proof. Let g^{L} be the number of guessed bits that correspond to the first κ bits of the encoding (i.e. the term $\mathsf{Bits}(\beta - \langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma} \rangle)$), and let g^{R} be the number of guessed bits that correspond to the remaining bits (i.e. the term $\boldsymbol{\gamma}$), such that $g = g^{\mathsf{L}} + g^{\mathsf{R}}$. Since $\boldsymbol{\gamma}$ is chosen uniformly, the probability that any g^{R} bits of $\boldsymbol{\gamma}$ can be guessed correctly is $2^{-g^{\mathsf{R}}}$. Impagliazzo and Naor [53, Proposition 1.1] show via the random choice of $\boldsymbol{\gamma}$ and the Leftover Hash Lemma [65] that the inner product $\langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma} \rangle$ has a statistical distance of at most $2^{(\kappa - |\boldsymbol{\gamma}|)/2}$ with respect to the uniform distribution. We wish to know the probability that g^{L} bits of this inner product can be guessed correctly, conditioned on g^{R} bits of $\boldsymbol{\gamma}$ being guessed correctly. Guessing g^{R} bits of $\boldsymbol{\gamma}$ correctly is equivalent to removing them (and their corresponding elements in $\mathbf{g}^{\mathsf{R}}$) from the inner product; thus under this condition $\langle \mathbf{g}^{\mathsf{R}}, \boldsymbol{\gamma} \rangle$ has a statistical distance of at most $2^{(\kappa - |\boldsymbol{\gamma}| + g^{\mathsf{R}})/2}$ with respect to the uniform distribution, and g^{L} bits of the inner product can be guessed with probability at most $2^{-g^{\mathsf{L}}} + 2^{(\kappa - |\boldsymbol{\gamma}| + g^{\mathsf{R}})/2}$. It follows that the probability of guessing g^{L} bits from the inner product and g^{R} bits from $\boldsymbol{\gamma}$ simultaneously is at most

$$2^{-g^{\mathsf{R}}} \cdot \left(2^{-g^{\mathsf{L}}} + 2^{(\kappa - |\boldsymbol{\gamma}| + g^{\mathsf{R}})/2}\right)$$

and substituting $|\boldsymbol{\gamma}| = \kappa + 2s$, we have

$$2^{-g^{\mathsf{L}} - g^{\mathsf{R}}} + 2^{-s - g^{\mathsf{R}}/2} \leq 2^{-g} + 2^{-s} \qquad \square$$

Hybrid $\boldsymbol{\mathcal{H}_1}$. This experiment is the same as a real-world execution of π_{Mul} except that Step 4 of $\mathcal{S}^{\mathsf{A}}_{\mathsf{Mul}}$ is implemented to define $\boldsymbol{\Delta}$, $\tilde{\boldsymbol{\omega}}$, and $\mathbf{I}$, and the experiment aborts with probability $1 - 2^{-|\mathbf{I}|}$ rather than aborting based on Step 6 of π_{Mul}. The addition of Step 4 of the simulator changes no variables in Alice's view, and is used only to define the aforementioned variables. We now argue that the abort events occurs with almost the same probability in both $\mathcal{H}_1$ and the real world execution of π_{Mul}.

In the real experiment, Alice induces Bob to abort when, after we implement Step 4 of the simulator, there exists some $i \in \mathbf{I}$ such that $\tilde{\boldsymbol{\omega}}_i$ is defined and $\tilde{\boldsymbol{\omega}}_i \neq \boldsymbol{\omega}_i$. In the trivial case, when $\tilde{\boldsymbol{\omega}}_i \notin \{0, 1\}$, Bob aborts with certainty. Otherwise, for $i \in \mathbf{I}$, we have

$$\begin{aligned}
\mathbf{r}_i &= \chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i} + \tilde{\boldsymbol{\omega}}_i \cdot \boldsymbol{\Delta}_i \\
\chi \cdot \mathbf{t}_{\mathsf{B}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{B}i} &= \boldsymbol{\omega}_i \cdot (\chi \cdot \boldsymbol{\alpha} + \hat{\chi} \cdot \hat{\boldsymbol{\alpha}}_i) - (\chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i}) \\
&= (1 - \tilde{\boldsymbol{\omega}}_i)(u + \boldsymbol{\Delta}_i) - (\chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i}) \\
&= \boldsymbol{\omega}_i \cdot u + \boldsymbol{\Delta}_i - (\chi \cdot \mathbf{t}_{\mathsf{A}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{A}i} + \tilde{\boldsymbol{\omega}}_i \boldsymbol{\Delta}_i) \\
&= \boldsymbol{\omega}_i \cdot u + \boldsymbol{\Delta}_i - \mathbf{r}_i \\
&\neq \boldsymbol{\omega}_i \cdot u - \mathbf{r}_i
\end{aligned}$$

where the last line follows because $\boldsymbol{\Delta}_i \neq 0$ when $\tilde{\boldsymbol{\omega}}_i$ is defined. Thus if Alice cheats in such a way that, for any $i \in \mathbf{I}$, $\tilde{\boldsymbol{\omega}}_i \neq \boldsymbol{\omega}_i$, then Bob aborts, but if $\tilde{\boldsymbol{\omega}}_i = \boldsymbol{\omega}_i$ for all $i \in \mathbf{I}$, then she cheats and is uncaught. Consequently, the probability that Alice can cheat without being caught in the real world is the same as the probability that she correctly guesses Bob's input $\boldsymbol{\omega}_i$ to $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ at every location $i \in \mathbf{I}$ where she has cheated. By Lemma E.3, this probability is at most $2^{-|\mathbf{I}|} + 2^{-s}$. On the other hand, she sees an abort in $\mathcal{H}_1$ with probability $2^{-|\mathbf{I}|}$. The advantage of a distinguisher in distinguishing this hybrid from the real protocol is therefore at most 2^{-s}.

Hybrid $\boldsymbol{\mathcal{H}_2}$. This experiment is the same as $\mathcal{H}_1$, except the following instruction is added after Alice sends her second message: Find any index $i \in [1, \ell]$ such that $\chi \cdot \boldsymbol{\alpha}_i + \hat{\chi} \cdot \hat{\boldsymbol{\alpha}}_i = u$, and set $\alpha := \boldsymbol{\alpha}_i$. If no such index exists, or there exist two indices i, j for which the condition holds, but $\boldsymbol{\alpha}_i \neq \boldsymbol{\alpha}_j$, then abort.

This hybrid experiment differs from the last in that it aborts if there is not exactly one unique candidate for α, and thus a malicious Alice can distinguish $\mathcal{H}_2$ from $\mathcal{H}_1$ by inducing such a scenario without causing an abort. We argue that this event occurs with probability $\text{poly}(\kappa)/2^{\kappa}$ by analyzing both conditions (no candidate, or too many candidates) and taking a union bound.

Consider the event in $\mathcal{H}_1$ when there is no candidate pair $(\boldsymbol{\alpha}_i, \hat{\boldsymbol{\alpha}}_i)$ such that $\chi \cdot \boldsymbol{\alpha}_i + \hat{\chi} \cdot \hat{\boldsymbol{\alpha}}_i = u$, and yet no abort has occurred. This implies that $\boldsymbol{\Delta}_i \neq 0$ (that is, Alice has cheated at location i) for *all* indices $i \in [1, \ell]$. This event occurs with probability $2^{-\ell}$.

Next, consider the event in $\mathcal{H}_1$ that there are two candidate pairs, with indices i and j. This implies that

$$\chi \cdot (\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_j) + \hat{\chi} \cdot (\hat{\boldsymbol{\alpha}}_i - \hat{\boldsymbol{\alpha}}_j) = 0$$

Note that $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\alpha}}$ are fixed before $\chi, \hat{\chi}$ are chosen, and that for each selection of χ, there is only one $\hat{\chi} \in \mathbb{Z}_q$ that satisfies the equality. Because $\chi, \hat{\chi}$ are chosen by hashing the transcript in Step 4 of π_{Mul}, a malicious Alice may attempt to produce different transcripts in order to satisfy this equality. Each random oracle query made by Alice succeeds in satisfying the condition with probability $2^{-\kappa}$; thus, Alice succeeds with probability no greater than $\text{poly}(\kappa)/2^{\kappa}$.

Hybrid $\mathcal{H}_3$. This hybrid is the same as the previous one, except that instead of using Step 7 of π_{Mul} to define Bob's output of the computation t_{B}, define Bob's output as

$$t_{\mathsf{B}} := \alpha \cdot \beta + \delta - t_{\mathsf{A}}$$

where

$$t_{\mathsf{A}} := \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \mathbf{t}_{\mathsf{A}i}$$
$$\delta := \sum_{i \in \mathbf{I}} \tilde{\boldsymbol{\omega}}_i \cdot \mathbf{g}_i \cdot (\boldsymbol{\alpha}_i - \alpha)$$

and where α, $\tilde{\boldsymbol{\omega}}$, and $\mathbf{I}$ are defined as in $\mathcal{H}_2$. Note that, as before, if $\tilde{\boldsymbol{\omega}}_i \neq \boldsymbol{\omega}_i$ for any $i \in \mathbf{I}$, then an abort occurs. This essentially implements Step 5 of $\mathcal{S}^{\mathsf{A}}_{\mathsf{Mul}}$.

Recall that $\mathbf{t}_{\mathsf{A}i} + \mathbf{t}_{\mathsf{B}i} = \boldsymbol{\alpha}_i \cdot \boldsymbol{\omega}_i$ and that the final output shares t_{A} and t_{B} maintain the following relation

$$t_{\mathsf{A}} + t_{\mathsf{B}} = \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \mathbf{t}_{\mathsf{A}i} + \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \mathbf{t}_{\mathsf{B}i} = \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \boldsymbol{\alpha}_i \cdot \boldsymbol{\omega}_i$$

Thus, if no abort occurs, then we have

$$\begin{aligned} t_{\mathsf{A}} + t_{\mathsf{B}} &= \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \boldsymbol{\alpha}_i \cdot \boldsymbol{\omega}_i \\ &= \alpha \cdot \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \boldsymbol{\omega}_i + \sum_{i \in \mathbf{I}} \mathbf{g}_i \cdot \tilde{\boldsymbol{\omega}}_i \cdot (\boldsymbol{\alpha}_i - \alpha) \\ &= \alpha \cdot \beta + \delta \end{aligned}$$

which is the relation claimed at the beginning of this hybrid. This hybrid is therefore distributed identically to $\mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. Implement the remaining steps in $\mathcal{S}^{\mathsf{A}}_{\mathsf{Mul}}$. These changes are merely syntactic, and thus Alice's view in $\mathcal{H}_4$ is identical to her view in $\mathcal{H}_3$.

The view produced by $\mathcal{S}^{\mathsf{A}}_{\mathsf{Mul}}$ is therefore indistinguishable from a real execution to a probabilistic polynomial time adversary corrupting Alice, with overwhelming probability. ☐

B. Simulating Against Bob

Simulator 4. Multiplication against Bob $\left(\mathcal{S}^{\mathsf{B}}_{\mathsf{Mul}}\right)$:

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

Init:

1) Receive message $(\texttt{init})$ from Bob on behalf of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ and send $(\texttt{init})$ to $\mathcal{F}_{\mathsf{Mul}}$.
2) Send $(\texttt{init-complete})$ to Bob on behalf of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ upon receipt of $(\texttt{init-complete})$ from $\mathcal{F}_{\mathsf{Mul}}$.

Multiplication:

3) Receive $(\texttt{choose}, \mathsf{id}^{\mathsf{mul}}, \boldsymbol{\omega})$ from Bob on behalf of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ and compute
$$\beta := \sum_{i \in [1,\ell]} \mathbf{g}_i \cdot \boldsymbol{\omega}_i$$
4) Send $(\texttt{input}, \mathsf{id}^{\mathsf{mul}}, \beta)$ to $\mathcal{F}_{\mathsf{Mul}}$
5) On receipt of $(\texttt{output}, \mathsf{id}^{\mathsf{mul}}, t_{\mathsf{B}})$ from $\mathcal{F}_{\mathsf{Mul}}$, uniformly sample $\hat{\mathbf{t}}_{\mathsf{B}}, \mathbf{t}_{\mathsf{B}} \leftarrow \mathbb{Z}_q^{\ell}$ such that
$$\sum_{i \in [1,\ell]} \mathbf{t}_{\mathsf{B}i} = t_{\mathsf{B}}$$
6) Send $(\texttt{padded-correlation}, \mathsf{id}^{\mathsf{mul}}, \{\mathbf{t}_{\mathsf{B}i} || \hat{\mathbf{t}}_{\mathsf{B}i}\}_{i \in [1,\ell]})$ to Bob on behalf of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$.
7) Engage in the coin flipping protocol (corresponding to Step 4 of π_{Mul}) to compute χ and $\hat{\chi}$ as Alice would.
8) Sample $u \leftarrow \mathbb{Z}_q$, and for each $i \in [1, \ell]$, compute
$$\mathbf{r}_i := \chi \cdot \mathbf{t}_{\mathsf{B}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{B}i} - \boldsymbol{\omega}_i \cdot u$$
9) Send Bob the values u and $\mathbf{r}$ and halt.

Lemma E.4. *The view produced by $\mathcal{S}^{\mathsf{B}}_{\mathsf{Mul}}$ and the view of a malicious Bob in a real execution of the protocol π_{Mul} are distributed identically in the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$-hybrid Random Oracle Model.*

Proof. The information in Bob's view is characterized by the outputs $\mathbf{t}_{\mathsf{B}}$ and $\hat{\mathbf{t}}_{\mathsf{B}}$ that he receives from $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ upon sending $\boldsymbol{\omega}$, and by the messages u and $\mathbf{r}$ that he receives from Alice.

As per the description of the $\mathcal{F}^{\ell}_{\mathsf{COTe}}$ functionality, its outputs maintain the following relations for all $i \in [1, \ell]$

$$\mathbf{t}_{\mathsf{A}i} + \mathbf{t}_{\mathsf{B}i} = \boldsymbol{\omega}_i \cdot \alpha \qquad \text{and} \qquad \hat{\mathbf{t}}_{\mathsf{A}i} + \hat{\mathbf{t}}_{\mathsf{B}i} = \boldsymbol{\omega}_i \cdot \hat{\alpha}$$

In the real world, $\mathbf{t}_{\mathsf{A}}$ and $\hat{\mathbf{t}}_{\mathsf{A}}$ are chosen uniformly at random by $\mathcal{F}^{\ell}_{\mathsf{COTe}}$, and thus $\mathbf{t}_{\mathsf{B}}$ and $\hat{\mathbf{t}}_{\mathsf{B}}$ are distributed uniformly as well. In the simulation, on the other hand, t_{B} is chosen uniformly by $\mathcal{F}_{\mathsf{Mul}}$, and then $\mathbf{t}_{\mathsf{B}}$ and $\hat{\mathbf{t}}_{\mathsf{B}}$ are chosen uniformly by the simulator, subject to

$$\sum_{i \in [1,\ell]} \mathbf{t}_{\mathsf{B}i} = t_{\mathsf{B}}$$

Thus, by the relationships above, the values $\mathbf{t}_{\mathsf{A}}$ and $\hat{\mathbf{t}}_{\mathsf{A}}$ are also distributed uniformly in the simulation, and all of these values are distributed identically in the real and simulated views.

In both the real and simulated views, $\hat{\alpha}$ is chosen uniformly at random, and χ and $\hat{\chi}$ are derived from a coin flipping protocol. Consequently, $u = \chi \cdot \alpha + \hat{\chi} \cdot \hat{\alpha}$ is distributed identically in both views. Finally, by correctness of the protocol, in the real world $\mathbf{r}$ is consistent with the following relation, for all $i \in [1, \ell]$

$$\mathbf{r}_i + \boldsymbol{\omega}_i \cdot u = \chi \cdot \mathbf{t}_{\mathsf{B}i} + \hat{\chi} \cdot \hat{\mathbf{t}}_{\mathsf{B}i}$$

In the simulation, the value $\boldsymbol{\omega}$ is received as on behalf of $\mathcal{F}^{\ell}_{\mathsf{COTe}}$, u is chosen by the simulator, $\chi, \hat{\chi}$ are public, and $\mathbf{t}_{\mathsf{B}}$ and $\hat{\mathbf{t}}_{\mathsf{B}}$ are chosen by the simulator. The simulator can therefore solve for $\mathbf{r}$, and given that the distribution of the former values is identical to their distribution in the real world, the resulting distribution of $\mathbf{r}$ must be identical to the real-world distribution as well. The views produced by real and simulated executions of π_{Mul} are therefore distributed identically to an adversary that corrupts Bob. □

Appendix F
Proof of Security for 2-of-n ECDSA

In this section, we prove the security of our setup and signing protocols via reductions to the Computational Diffie-Hellman Assumption and to the security of ECDSA itself in the Random Oracle Model. 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 which orchestrates a signing *epoch*, in which the parties perform a single setup as a group, followed by some number of signatures between pre-determined pairs. It is with respect to this shell protocol that we claim security.

Protocol 10. 2-of-n ECDSA $\left(\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}\right)$:

This protocol runs among a group of parties $\mathcal{P}$ of size n, where each party's unique player index in $[1, n]$ is known to the other parties. It is parameterized by the union of the parameters of its subprotocols; specifically, by the group $\mathbb{G}$ of order q generated by G, with $\kappa = |q|$, and by the statistical security paramter s. It receives as input a vector $\mathbf{m} \in \{0,1\}^{*\times *}$ of messages and a vector $\mathbf{P} \in \mathcal{P}^{2\times|\mathbf{m}|}$ of pairs of parties to sign those messages, such that in each pair the party with the smallest index always comes first. To each party, it outputs a vector of signatures. After the Setup phase, the Sign phase can be run repeatedly by pairs of parties from this group.

Setup (2-of-n):

1) The parties jointly run $\pi^{\mathsf{2P\text{-}Setup}}_{n\mathsf{P\text{-}ECDSA}}$ 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 entry $\mathbf{m}_j$ in $\mathbf{m}$, let $(\mathcal{P}_{\mathsf{A}}, \mathcal{P}_{\mathsf{B}}) = \mathbf{P}_j$. Now $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$ (that is, Alice and Bob) run $\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}$, supplying $p(\mathsf{A})$ and $p(\mathsf{B})$ respectively, and both supplying $\mathbf{m}_j$. $\mathcal{P}_{\mathsf{B}}$ receives the signature as output, and $\mathcal{P}_{\mathsf{A}}$ receives nothing.

Theorem F.1. *Assuming that ECDSA is a Digital Signature Scheme and that the Computational Diffie-Hellman Problem is hard in elliptic curve groups, the protocol $\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}$ UC-realizes the $\mathcal{F}_{\mathsf{SampledECDSA}}$ functionality among n parties in the $(\mathcal{F}_{\mathsf{Mul}}, \mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}, \mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}})$-hybrid Random Oracle Model and in the presence of a single statically corrupted malicious party.*

Proof. Our proof will be via a sequence of hybrid experiments showing that the view of a corrupted party $\mathcal{P}^*_i$ when participating in the real-world protocol $\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}$ is computationally indistinguishable from a view generated by the simulator $\mathcal{S}^{\mathsf{2P\text{-}Epoch},i}_{n\mathsf{P\text{-}ECDSA}}$, which accesses $\mathcal{F}_{\mathsf{SampledECDSA}}$. Specifically, let $\mathcal{Z}$ denote the *environment*, which chooses the messages to be signed, the pairs of parties that will sign them, and the order in which signing will happen. The environment is treated as an adversary: it receives some nonuniform advice z and it can corrupt exactly one party, $\mathcal{P}^*_i$, from whom it receives a transcript of the $\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}$ protocol. In a *real-world* experiment, the honest parties (i.e. all parties except $\mathcal{P}^*_i$) also evaluate the $\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}$ protocol, using the inputs chosen by $\mathcal{Z}$. In an *ideal-world* experiment, the honest parties instead interact with $\mathcal{F}_{\mathsf{SampledECDSA}}$, while $\mathcal{P}^*_i$ interacts with $\mathcal{S}^{\mathsf{2P\text{-}Epoch},i}_{n\mathsf{P\text{-}ECDSA}}$, which in turn interacts with $\mathcal{F}_{\mathsf{SampledECDSA}}$ as an ideal-world adversary. When all interactions are complete, $\mathcal{Z}$ guesses whether $\mathcal{P}^*_i$ has interacted with real counterparties, or with the simulator; this guess is the output of the experiment. If we denote by $\mathsf{EXPT}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ the outcome of the experiment EXPT involving the protocol π, the adversary $\mathcal{A}$, and the environment $\mathcal{Z}$ which receives advice z, then the space of real-world experiments is given by

$$\mathcal{H}_0 = \left\{\mathsf{REAL}_{\pi^{\mathsf{2P\text{-}Epoch}}_{n\mathsf{P\text{-}ECDSA}}, \mathcal{P}^*_i, \mathcal{Z}}(z)\right\}_{z \in \{0,1\}^*}$$

We wish to show that

$$\mathcal{H}_0 \overset{c}{\equiv} \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SampledECDSA}}, \mathcal{S}^{\mathsf{2P\text{-}Epoch},i}_{n\mathsf{P\text{-}ECDSA}}, \mathcal{Z}}(z)\right\}_{z \in \{0,1\}^*}$$

Due to the length and complexity of this proof, we have divided it into sections. In Appendix F-A, we give a hybrid in which the components of $\mathcal{P}^*_i$'s transcript that are due to the setup protocol $\pi^{\mathsf{2P\text{-}Setup}}_{n\mathsf{P\text{-}ECDSA}}$ are simulated. In Appendix F-B, we give a further sequence of hybrids that replace the transcript components due to cases in which $\mathcal{P}^*_i$ participates in the signing protocol $\pi^{\text{2P-Sign}}_{n\text{P-ECDSA}}$ and plays the role of Alice. In Appendix F-C, we likewise give a sequence of hybrids to deal with cases in which $\mathcal{P}^*_i$ plays the role of Bob, at which point $\mathcal{P}^*_i$'s view is totally simulated, and the proof is complete. We begin by giving a master simulator, which corresponds to $\pi^{\text{2P-Epoch}}_{n\text{P-ECDSA}}$ and calls upon the simulators we introduce in subsequent sections.

Simulator 5. 2-of-n ECDSA against $\mathcal{P}^*_i$ ($\mathcal{S}^{\text{2P-Epoch},i}_{n\text{P-ECDSA}}$):
This simulator interposes between a malicious $\mathcal{P}^*_i$ and the ideal functionality $\mathcal{F}_{\text{SampledECDSA}}$. It is parameterized by the union of the parameters of the simulators that it calls; specifically, by the group $\mathbb{G}$ of order q generated by G, with $\kappa = |q|$, and by the statistical security paramter s. It receives as input a vector of messages $\mathbf{m} \in \{0,1\}^{*\times *}$ and a vector of counterparties $\mathbf{P} \in \mathcal{P}^{|\mathbf{m}|}$ with which to sign those messages.
Setup (2-of-n):
1) Run $\mathcal{S}^{\text{Setup},i}_{n\text{P-ECDSA}}$ against $\mathcal{P}^*_i$ and receive the public key pk and $\mathcal{P}^*_i$'s point $p(i)$.

Signing:
2) For each item $\mathbf{m}_j$ in $\mathbf{m}$, use the counterparty index $\mathbf{P}_j$ to calculate the appropriate Lagrange coefficient, and use this to reconstruct the secret key share t^0_i for signing with $\mathbf{P}_j$. If $\mathbf{P}_j > i$, invoke $\mathcal{S}^{\text{2P-Sign},\text{A}}_{n\text{P-ECDSA}}$ against $\mathcal{P}^*_i$ with message $\mathbf{m}_j$, secret key share t^0_i, public key pk, and a fresh signature id $\mathsf{id}^{\mathsf{sig}}$. Otherwise, invoke $\mathcal{S}^{\text{2P-Sign},\text{B}}_{n\text{P-ECDSA}}$ with the same inputs.

A. Simulating Setup

We begin by giving a simulator $\mathcal{S}^{\text{Setup},i}_{n\text{P-ECDSA}}$, which interacts with $\mathcal{F}_{\text{SampledECDSA}}$ and the corrupt party $\mathcal{P}^*_i$, and produces a transcript corresponding to an invocation of $\pi^{\text{2P-Setup}}_{n\text{P-ECDSA}}$.

Simulator 6. 2-of-n Setup against $\mathcal{P}^*_i$ ($\mathcal{S}^{\text{Setup},i}_{n\text{P-ECDSA}}$):
This simulator interposes between a malicious $\mathcal{P}^*_i$ and the ideal functionality $\mathcal{F}_{\text{SampledECDSA}}$, and it is parameterized by the group $\mathbb{G}$ of order q generated by G, with $\kappa = |q|$. It returns as output $\mathcal{P}^*_i$'s point $p(i)$ on a shared polynomial p, and the public key pk. It makes use of the functionalities $\mathcal{F}^{R_{\text{DL}}}_{\text{ZK}}$ and $\mathcal{F}_{\text{Mul}}$.
Public Key Generation:
1) Send $(\texttt{init})$ to $\mathcal{F}_{\text{SampledECDSA}}$ and receive $(\texttt{public-key}, \mathsf{pk})$.
2) Receive $(\texttt{com-proof}, \mathsf{sk}_i, G')$ from $\mathcal{P}^*_i$ on behalf of $\mathcal{F}^{R_{\text{DL}}}_{\text{Com-ZK}}$ and receive pk_i from $\mathcal{P}^*_i$ directly. Send $\{(\texttt{committed}, j)\}_{j\in[1,i)\cup(i,n]}$ to $\mathcal{P}^*_i$ on behalf of $\mathcal{F}^{R_{\text{DL}}}_{\text{Com-ZK}}$. If $\mathsf{sk}_i \cdot G' \neq \mathsf{pk}_i$, then choose the parties' public key fragments to be a set of uniform values
$$\left\{\mathsf{pk}_j\right\}_{j\in[1,i)\cup(i,n]} \leftarrow \mathbb{G}^{n-1}$$
Otherwise, set them to be a random $n-1$ element additive sharing of $(\mathsf{pk} - \mathsf{pk}_i)$. That is, for $j \in [1,i)\cup(i,n]$, sample pk_j uniformly from $\mathbb{G}$ subject to
$$\sum_{j\in[1,i)\cup(i,n]} \mathsf{pk}_j = (\mathsf{pk} - \mathsf{pk}_i)$$
Send $\{(\texttt{accept}, j, \mathsf{pk}_j)\}_{j\in[1,i)\cup(i,n]}$ to $\mathcal{P}^*_i$ on behalf of $\mathcal{F}^{R_{\text{DL}}}_{\text{ZK}}$ and receive $(\texttt{decom-proof})$ from $\mathcal{P}^*_i$ on behalf of $\mathcal{F}^{R_{\text{DL}}}_{\text{ZK}}$. If $\mathsf{sk}_i \cdot G' \neq \mathsf{pk}_i$, then abort.
3) For $j \in [1,i)\cup(i,n]$, sample $p_j(i) \leftarrow \mathbb{Z}_q$ and send it to $\mathcal{P}^*_i$. Receive $p_{i,j}(j)$ from $\mathcal{P}^*_i$ in response.
4) Pick any $j \in [1,i)\cup(i,n]$ and interpolate $\mathcal{P}^*_i$'s polynomial p_i as the line that passes through the points $(0, \mathsf{sk}_i)$ and $\big(j, p_{i,j}(j)\big)$.
5) Enumerate any inconsistent shares that $\mathcal{P}_i$ may have sent, and calculate the offset by which they are incorrect:
$$\mathbf{\Delta} := \left\{p_i(j) - p_{i,j}(j)\right\}_{j\in[1,i)\cup(i,n]}$$
6) Compute the expected public commitment to $\mathcal{P}^*_i$'s share of the secret key
$$T^{\mathsf{exp}}_i := \sum_{j\in[1,n]} p_j(i)\cdot G$$
7) For $j \in [1,i)\cup(i,n]$, calculate the appropriate Lagrange coefficients $\lambda_{i,j}, \lambda_{j,i}$ for Shamir-reconstruction between $\mathcal{P}^*_i$ and $\mathcal{P}_j$, and then compute
$$T_j := \frac{\big(\mathsf{pk} - \lambda_{i,j}\cdot T^{\mathsf{exp}}_i\big)}{\lambda_{j,i}} + \mathbf{\Delta}_j \cdot G$$
and send T_j to $\mathcal{P}^*_i$.
8) Receive T_i from $\mathcal{P}^*_i$, and abort if $T_i \neq T^{\mathsf{exp}}_i$ or if there exists any index j such that $\mathbf{\Delta}_j \neq 0$.

Auxiliary setup:
9) Interact with $\mathcal{P}^*_i$ on behalf of $\mathcal{F}_{\text{Mul}}$, receiving (init) messages and replying with $(\mathsf{init\text{-}complete})$ messages as appropriate.

Hybrid $\mathcal{H}_1$. This hybrid experiment is the same as $\mathcal{H}_0$ except that $\mathcal{P}^*_i$'s invocation of $\pi^{\text{2P-Setup}}_{n\text{P-ECDSA}}$ is replaced by an execution of $\mathcal{S}^{\text{Setup},i}_{n\text{P-ECDSA}}$. The simulation is perfect. Consider the case that $\mathcal{P}^*_i$ attempts to cheat by sending points along p_i that do not represent a line, or that represent a line that does not pass through sk_i. This implies that there must exist two parties j and j' such that $\lambda_{j,j'}\cdot p(j) + \lambda_{j',j}\cdot p(j') \neq \mathsf{sk}$. In the real world, it follows that $\lambda_{j,j'}\cdot T_j + \lambda_{j',j}\cdot T_{j'} \neq \mathsf{pk}$, and therefore the parties j and j' will abort. Because the simulator receives $p_{i,j}(j)$ and $p_{i,j'}(j')$, it can calculate the exact discrepancy that appears in the real world and induce it into T_j and $T_{j'}$, while aborting in exactly the correct cases. Note that because the values T_j and $T_{j'}$ are randomized, only the relationships between them need be simulated, and so we can choose an arbitrary point along p_i from which to calculate any discrepancies.

B. Simulating Against Alice

We now consider the view of $\mathcal{P}^*_i$ in its evaluations of $\pi^{\text{2P-Sign}}_{n\text{P-ECDSA}}$ with parties $\mathcal{P}_j$ where $j > i$, and show via the series of hybrids in this section that this view is indistinguishable from that generated by the simulator $\mathcal{S}^{\text{2P-Sign,A}}_{n\text{P-ECDSA}}$ that interacts with the ideal functionality $\mathcal{F}_{\text{SampledECDSA}}$. For any specific j, the roles played by $\mathcal{P}^*_i$ and $\mathcal{P}_j$ are consistent and determined only by their indices. Furthermore, the post-setup interactions of $\mathcal{P}^*_i$ and $\mathcal{P}_j$ are independent of $\mathcal{P}^*_i$'s interactions with any other honest parties. Consequently, we consider each honest $\mathcal{P}_j$ individually, and for each sequential instance of the signing protocol evaluated by $\mathcal{P}^*_i$ and $\mathcal{P}_j$ we apply the following hybrids to show that instance indistinguishable from a simulation. For readability and convenience, we refer to $\mathcal{P}^*_i$ as Alice in this section.

Before we specify the hybrids in this section, let us discuss the manner in which a malicious adversary playing the role of Alice can cheat, and the power that this cheating confers. Alice sends only one message to Bob, and her sole mechanism for cheating is the sending of inconsistent values in this message. Specifically, Alice has an instance key k_{A} which is defined by

$$k_{\mathsf{A}} = H(k'_{\mathsf{A}} \cdot D_{\mathsf{B}}) + k'_{\mathsf{A}}$$

where k'_{A} is chosen adversarially by her. She also has a secret key share t^0_{A}, from which pk was calculated, and a pad ϕ, which she chooses uniformly. The values she uses in the protocol, however, are R' and the multiplication input vectors $\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^{2\text{a}}$, and $\boldsymbol{\alpha}^{2\text{b}}$, all of which are functions of t^0_{A}, k_{A}, and ϕ. We use a proof of knowledge to verify the relationship between R' and k_{A}, but in the case of the multiplication inputs, cheating on her part will introduce additive offsets into the outputs of the multiplication protocol that depend directly on Bob's private inputs. Any offset in the final signature will cause Bob to abort, which Alice can avoid by subtracting an equal offset from her final message: in this way, she can guess Bob's input and confirm that she is correct. We show that she is unable to do this with non-negligible probability unless she has the power to break the Computational Diffie-Hellman or Discrete Log Assumptions for Elliptice Curves. We use δ^1, $\delta^{2\text{a}}$, and $\delta^{2\text{b}}$ to signify the offsets Alice induces by cheating in $\mathcal{F}_{\mathsf{Mul}}$, and we use δ^3, δ^4, δ^5, and δ^{sig} to signify the offsets she induces directly in her subsequent messages, which she can use to cancel out the previous offsets. The propagation of these values through Alice and Bob's calculations in the real world is illustrated in Figure 3.

Simulator 7. 2-of-n Signing against Alice ($\mathcal{S}^{\text{2P-Sign,A}}_{n\text{P-ECDSA}}$):

This simulator interposes between a malicous Alice and the corresponding ideal functionality $\mathcal{F}_{\text{SampledECDSA}}$. It receives as input a message m, the signature id $\mathsf{id}^{\mathsf{sig}}$, the public key pk, and Alice's share of the secret key t^0_{A}. It is parameterized by statistical security parameter s, and the group $\mathbb{G}$ of order q generated by G, with $\kappa = |q|$. It plays the roles of the functionalities $\mathcal{F}_{\mathsf{Mul}}$ and $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$ in their interactions with Alice, and can both observe Alice's queries to the random oracle H and program its responses.

Multiplication and Instance Key Exchange:

1) Send $(\texttt{newsig}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{B})$ to $\mathcal{F}_{\text{SampledECDSA}}$ to begin a new signature with Bob as the counterparty, and receive $(\texttt{nonce-shard}, D_{\mathsf{B}})$ in response. Forward D_{B} to Alice.
2) Observe Alice's queries to the random oracle H, and let her i^{th} query be R_i. If R_i has never been queried before, send $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i)$ to $\mathcal{F}_{\text{SampledECDSA}}$, receive $(\texttt{offset}, \mathsf{id}^{\mathsf{sig}}, k^{\Delta}_{i,A})$ in response, and store $(\mathsf{id}^{\mathsf{sig}}, R_i, k^{\Delta}_{i,A})$ in memory. Program H to return $k^{\Delta}_{i,A}$ to Alice as the result of her query on R_i.
3) Upon receiving R' from Alice, find $(\mathsf{id}^{\mathsf{sig}}, R_{\mathrm{j}}, k^{\Delta}_{\mathrm{j},A})$ in memory such that $R_{\mathrm{j}} = R'$. If such an entry exists in memory, remember the index j and set $R \coloneqq R' + k^{\Delta}_{\mathrm{j},A} \cdot D_{\mathsf{B}}$; otherwise, choose any value for j, let $R_{\mathrm{j}} \coloneqq R'$, and abort by skipping to Step 8.
4) Interact with Alice on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$. On receiving $(\texttt{prove}, k_{\mathsf{A}}, D_{\mathsf{B}})$ from her, if $k_{\mathsf{A}} \cdot D_{\mathsf{B}} \neq R$, then abort by skipping to Step 8.
5) Interact with Alice on behalf of $\mathcal{F}_{\mathsf{Mul}}$, and in doing so receive her multiplication inputs, $\left(\alpha^1, \delta^1, c^1\right)$, $\left(\alpha^{2\text{a}}, \delta^{2\text{a}}, c^{2\text{a}}\right)$, and $\left(\alpha^{2\text{b}}, \delta^{2\text{b}}, c^{2\text{b}}\right)$, and her (uniform) output shares t^1_{A}, $t^{2\text{a}}_{\mathsf{A}}$, and $t^{2\text{b}}_{\mathsf{A}}$. Toss $c^1 + c^{2\text{a}} + c^{2\text{b}}$ coins, and if any of these coins return 1, or if $\alpha^{2\text{a}} \neq t^0_{\mathsf{A}}/k_{\mathsf{A}}$, or if $\alpha^{2\text{b}} \neq 1/k_{\mathsf{A}}$, then abort by skipping to Step 8. Otherwise, compute and store $\phi = \alpha^1 - 1/k_{\mathsf{A}}$.

Consistency Check, Signature, and Verification:

6) Upon receiving η^{ϕ} from Alice, observe Alice's queries to the random oracle H, and denote her i^{th} query as Γ^1_i. For each query on Γ^1_i, check whether

$$\phi \stackrel{?}{=} \eta^{\phi} - H(\Gamma^1_i)$$
$$\Gamma^1_i \stackrel{?}{=} G + \phi \cdot k_{\mathsf{A}} \cdot G - t^1_{\mathsf{A}} \cdot R + \delta^1 \cdot R$$

If no query exists such that these relations hold, abort by skipping to Step 8. Otherwise, let $\Gamma^1 \coloneqq \Gamma^1_i$.
7) Upon receiving η^{sig} from Alice, compute

$$\delta^2 \coloneqq \delta^{2a} + \delta^{2b}$$
$$\Gamma^2 \coloneqq \left(t^1_{\mathsf{A}} - \delta^1\right) \cdot \mathsf{pk} - \left(t^2_{\mathsf{A}} - \delta^2\right) \cdot G$$
$$\mathsf{sig}_{\mathsf{A}} \coloneqq \eta^{\mathsf{sig}} - H(\Gamma^2)$$

Finally, if

$$\mathsf{sig}_{\mathsf{A}} \neq H(m) \cdot t^1_{\mathsf{A}} + r_x \cdot t^2_{\mathsf{A}} - H(m) \cdot \delta^1 - r_x \cdot \delta^2$$

then abort by skipping to Step 8. Otherwise send $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, \mathrm{j}, k_{\mathsf{A}})$ to $\mathcal{F}_{\text{SampledECDSA}}$ and halt *without* proceeding to Step 8.

Abort:

8) Cause $\mathcal{F}_{\text{SampledECDSA}}$ to abort by choosing any k^*_{A} such that $k^*_{\mathsf{A}} \cdot D_{\mathsf{B}} \neq R'$, sending $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, \mathrm{j}, k^*_{\mathsf{A}})$ to $\mathcal{F}_{\text{SampledECDSA}}$, and halting. Note that this step is only reached when Alice has cheated.

Hybrid $\mathcal{H}_2$. This hybrid experiment implements Steps 3 and 4 of $\mathcal{S}^{\text{2P-Sign,A}}_{n\text{P-ECDSA}}$. It is the same as $\mathcal{H}_1$, except that it aborts when Alice sends a value of R' to Bob for which she has not queried to the random oracle. When she does send such an R', $\mathcal{H}_1$ also aborts with overwhelming probability, since k_{A}, her input to

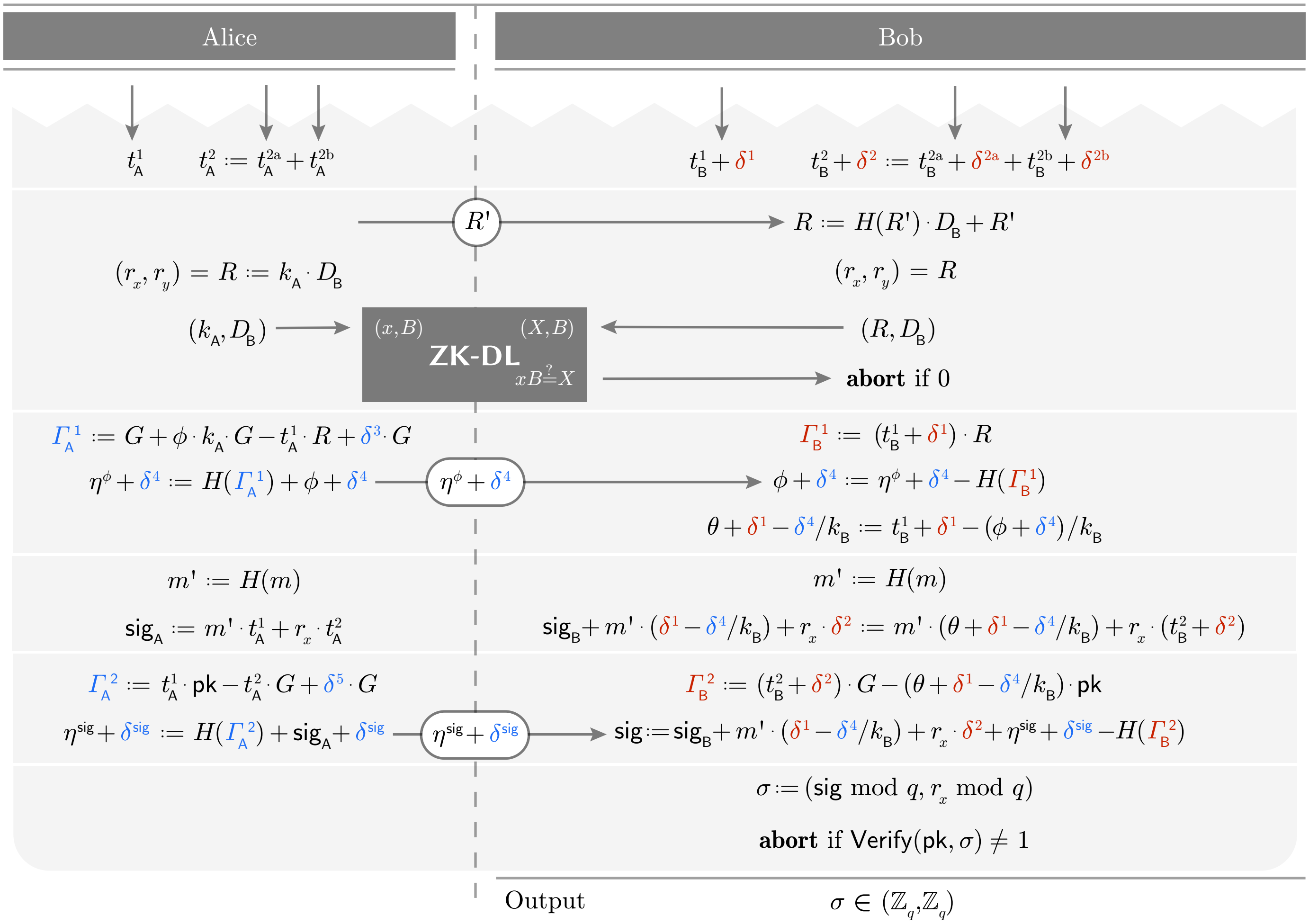

Fig. 3: **Illustrated Propagation of Noise and Offsets for a Malicious Alice in our 2-of-n Signing Scheme**. Noise induced by inconsistencies in the multiplication input vectors α^1, α^{2a}, and α^{2b} is indicated with red text, and offsets chosen by Alice are indicated with blue text. Note that both noise and offsets are defined relative to the ideal values of other variables derived from Alice's canonical inputs.

the proof of knowledge, is derived from the oracle's response. Alice can guess the correct value of k_A with probability no greater than $2^{-\kappa}$ without querying the random oracle, and thus her statistical advantage in distinguishing $\mathcal{H}_2$ from $\mathcal{H}_1$ is $2^{-\kappa}$.

Hybrid $\mathcal{H}_3$. This hybrid experiment is identical to $\mathcal{H}_2$, except that it aborts if Alice does not query the random oracle on the value $\Gamma^1 = t_B^1 \cdot R$, where t_B^1 is computed as Bob would compute it, using the information in his view. In the real world, Bob makes an identical query in order to decrypt η^ϕ and η^{sig}. If Alice has not also queried the oracle on this point, she cannot have sent the correct encryptions except with probability $2^{-\kappa}$. Thus, $\mathcal{H}_3$ is distinguishable from $\mathcal{H}_2$ with probability $2^{-\kappa}$.

To make the next step, we require a lemma to show that an adversary who can compute G/x given $x \cdot G$ can be used to solve the Computational Diffie-Hellman Problem.

Lemma F.2. *Let q be the order of a group $\mathbb{G}$ generated by G, whose elements are represented in $\kappa = |q|$ bits. If there exists a PPT algorithm $\mathcal{A}$ such that*

$$\Pr[\mathcal{A}(1^\kappa, x \cdot G) = G/x : x \leftarrow \mathbb{Z}_q] = \varepsilon$$

where the probability is taken over the choice of x, then there exists an algorithm $\mathcal{A}'$ such that

$$\Pr[\mathcal{A}'(1^\kappa, x \cdot G) = x \cdot x \cdot G : x \leftarrow \mathbb{Z}_q] = \varepsilon^3$$

Proof. The algorithm that receives a challenge $X = x \cdot G$ and uses $\mathcal{A}$ to compute $x \cdot x \cdot G$ is as follows:

1) Sample $z \leftarrow \mathbb{Z}_q$ uniformly and compute $Y := (z \cdot G) - X$. Let y be the discrete log of Y. Note that y is unknown, but that $z = x + y$. As z is uniform and independent of X, so is y.
2) Compute $X' := \mathcal{A}(X)$ and $Y' := \mathcal{A}(Y)$. Recall that X and Y are uniform and independent challenges to $\mathcal{A}$. If $\mathcal{A}$ was successful in both computations, then $X' = G/x$ and $Y' = G/y$.
3) Sample $r \leftarrow \mathbb{Z}_q$ uniformly and compute

$$W := r \cdot \mathcal{A}\left(r/z \cdot (X' + Y')\right)$$

Note that since r is sampled independently of X and Y, the challenge given to $\mathcal{A}$ is uniform and independent of X and Y. Note that

$$X' + Y' = G/x + G/y = \frac{x+y}{x \cdot y} \cdot G = \frac{z}{x \cdot y} \cdot G$$

Thus, if $\mathcal{A}$ was successful in this step,

$$W = r \cdot \frac{z}{r} \cdot \frac{x \cdot y}{z} \cdot G = x \cdot y \cdot G = x \cdot z \cdot G - x \cdot x \cdot G$$

4) Output $z \cdot X - W = x \cdot x \cdot G$

The algorithm $\mathcal{A}$ is successful with probability ε when its input is uniformly distributed. As $\mathcal{A}$ is invoked with three uniform and independent challenges, the probability that it is correct all three times is ε^3. □

Corollary F.2.1. *Let q be the order of a group $\mathbb{G}$ generated by G, whose elements are represented in $\kappa = |q|$ bits. If there exists a PPT algorithm $\mathcal{A}$ such that*

$$\Pr[\mathcal{A}(1^\kappa, x \cdot G) = G/x : x \leftarrow \mathbb{Z}_q] = \varepsilon$$

where the probability is taken over the choice of x, then there exists an algorithm $\mathcal{A}'$ such that $\mathcal{A}'$ solves the Computational Diffie-Hellman problem in $\mathbb{G}$ with advantage ε^6.

Proof. Follows directly from Lemma F.2 and Lemma D.2. □

Hybrid $\mathcal{H}_4$. This hybrid experiment implements Step 5 of $\mathcal{S}^{\mathsf{2P\text{-}Sign},\mathsf{A}}_{n\mathsf{P\text{-}ECDSA}}$. It differs from $\mathcal{H}_3$ in that it always aborts when any of Alice's inputs to $\mathcal{F}_{\mathsf{Mul}}$ are inconsistent with one another or with the instance key exchange. Recall that her inputs are given by the triples $(\alpha^1, \delta^1, c^1)$, $(\alpha^{2a}, \delta^{2a}, c^{2a})$, and $(\alpha^{2b}, \delta^{2b}, c^{2b})$. Specifically, Alice sees an abort if any of the following conditions hold

$$\eta^\phi - H(\mathit{\Gamma}^1) \neq \alpha^1 - 1/k_{\mathsf{A}}$$

$$\alpha^{2a} \neq t^0_{\mathsf{A}}/k_{\mathsf{A}} \qquad \alpha^{2b} \neq 1/k_{\mathsf{A}}$$

On the other hand, in experiment $\mathcal{H}_3$, it is possible that some of these conditions hold and yet no abort occurs if Alice offsets her final message η^{sig} in such a way that Bob reconstructs a valid signature in Step 13 of $\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}$. We will show that if she can achieve this outcome (and thereby distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$) with non-negligible probability, then she can be used to solve either the Computational Diffie-Hellman Problem or the Discrete Logarithm Problem over elliptic curves with non-negligible probability. It thus follows that $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$ by the Computational Diffie-Hellman Assumption.

For convenience, we define a few intermediate values. Let

$$e^1 = \eta^\phi - H(\mathit{\Gamma}^1) - \alpha^1 + 1/k_{\mathsf{A}}$$
$$e^{2a} = \alpha^{2a} - t^0_{\mathsf{A}}/k_{\mathsf{A}}$$
$$e^{2b} = \alpha^{2b} - 1/k_{\mathsf{A}}$$

That is, let e^1 be the discrepancy between the value of ϕ that Bob calculates and the value that Alice has used in her input to $\mathcal{F}_{\mathsf{Mul}}$, and let e^{2a} and e^{2b} be the discrepancies between α^{2a} and α^{2b} and their respective ideal values. Note that e^{2a} and e^{2b} are defined exclusively by extractable values in Alice's view: t^0_{A} is extracted during setup, k_{A} is extracted via $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$, and the remaining α and t values are extracted via $\mathcal{F}_{\mathsf{Mul}}$. e^1, on the other hand, depends upon $\mathit{\Gamma}^1$, which is defined in this hybrid relative to values in Bob's view (though, per $\mathcal{H}_3$, Alice is guaranteed to query the random oracle on $\mathit{\Gamma}^1$). Our reductions will not have access to Bob's view, and therefore they will require an alternate means of determining $\mathit{\Gamma}^1$ and e^1.

In the case where Alice can potentially distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$, at least one of the error values e^1, e^{2a}, and e^{2b} must be non-zero. These are included with Alice's inputs to $\mathcal{F}_{\mathsf{Mul}}$, and the values δ^1, δ^{2a}, and δ^{2b} represent additional errors induced into the outputs of $\mathcal{F}_{\mathsf{Mul}}$; thus we have

$$t^1_{\mathsf{A}} + t^1_{\mathsf{B}} = \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{\phi + e^1}{k_{\mathsf{B}}} + \delta^1$$
$$t^1_{\mathsf{A}} + \theta = \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^1}{k_{\mathsf{B}}} + \delta^1$$
$$t^{2a}_{\mathsf{A}} + t^{2a}_{\mathsf{B}} = \frac{t^0_{\mathsf{A}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^{2a}}{k_{\mathsf{B}}} + \delta^{2a}$$
$$t^{2b}_{\mathsf{A}} + t^{2b}_{\mathsf{B}} = \frac{t^0_{\mathsf{B}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{t^0_{\mathsf{B}} \cdot e^{2b}}{k_{\mathsf{B}}} + \delta^{2b}$$

Additionally, Alice can induce an error directly into the signature by including it in her final message η^{sig}; this is given by

$$\delta^{\mathsf{sig}} = \mathsf{sig}_{\mathsf{A}} - H(m) \cdot t^1_{\mathsf{A}} - r_x \cdot t^2_{\mathsf{A}}$$

Adding this to our signing equation, substituting in the error-laden shares above, and subtracting the value of an honest signature, we arrive at the total error induced

$$H(m) \cdot \left(\frac{e^1}{k_{\mathsf{B}}} + \delta^1 \right) + r_x \cdot \left(\frac{e^{2a} + t^0_{\mathsf{B}} \cdot e^{2b}}{k_{\mathsf{B}}} + \delta^2 \right) + \delta^{\mathsf{sig}}$$

where $\delta^2 = \delta^{2a} + \delta^{2b}$. According to our premise that the signature verifies, this expression must be equal to zero. Assuming this to be true, and then rearranging, we have

$$H(m) \cdot \frac{e^1}{k_{\mathsf{B}}} + r_x \cdot \frac{e^{2a} + t^0_{\mathsf{B}} \cdot e^{2b}}{k_{\mathsf{B}}} = -H(m) \cdot \delta^1 - r_x \cdot \delta^2 - \delta^{\mathsf{sig}}$$

We now partition our argument into three exhaustive subcases, based upon the value of the left hand side of this equation.

1) The first case:

$$H(m) \cdot \frac{e^1}{k_{\mathsf{B}}} + r_x \cdot \frac{e^{2a} + t^0_{\mathsf{B}} \cdot e^{2b}}{k_{\mathsf{B}}} \neq 0$$

In this case, we give a reduction to the Discrete Logarithm Problem. Because both sides are nonzero, we can once again rearrange the equation that defines our error, yielding

$$k_{\mathsf{B}} = \frac{H(m) \cdot e^1 + r_x \cdot \left(e^{2a} + t^0_{\mathsf{B}} \cdot e^{2b}\right)}{-H(m) \cdot \delta^1 - r_x \cdot \delta^2 - \delta^{\mathsf{sig}}}$$

Thus, if Alice can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ when this case holds, then can be used to solve the Discrete Logarithm Problem with the same probability in the following way: given $X = x \cdot G$ for which x is not known, run hybrid experiment $\mathcal{H}_4$, choosing a value of sk uniformly and calculating pk from it, rather than receiving pk and from $\mathcal{F}_{\mathsf{SampledECDSA}}$, and setting $D_{\mathsf{B}} := X$. During the course of the experiment, all values in the right-hand side of the above equation are extracted or received from Alice, except for t^0_{B}, which can be calculated via $t^0_{\mathsf{B}} := \mathsf{sk} - t^0_{\mathsf{A}}$, and e^1, which depends on Γ^1 as noted previously. Since Alice is guaranteed to query the random oracle on Γ^1, iterate over all of her queries, and for each candidate Γ^1_i, calculate $k_{\mathsf{B}i}$ via the above equation. Upon finding the query index i such that $k_{\mathsf{B}i} \cdot G = X$, let $x := k_{\mathsf{B}i}$ and the problem is solved.

2) The second case:

$$H(m) \cdot \frac{e^1}{k_{\mathsf{B}}} + r_x \cdot \frac{e^{2\mathrm{a}} + t^0_{\mathsf{B}} \cdot e^{2\mathrm{b}}}{k_{\mathsf{B}}} = 0 \quad \wedge \quad e^{2\mathrm{b}} \neq 0$$

In this case we also give a reduction to the Discrete Logarithm Problem. Observe that by rearranging the premise of the case

$$t^0_{\mathsf{B}} = \frac{-H(m) \cdot e^1}{r_x \cdot e^{2\mathrm{b}}} - \frac{e^{2\mathrm{a}}}{e^{2\mathrm{b}}}$$

Thus, if Alice can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ when this case holds, then she can be used to solve the Discrete Logarithm Problem with the same probability in the following way: given $X = x \cdot G$, for which x is unknown, run hybrid experiment $\mathcal{H}_4$, choosing $\mathsf{pk} := X$. As in the first case, iterate over all of Alice's random oracle queries, and for each candidate point Γ^1_i, apply the above equation to calculate $t^0_{\mathsf{B}i}$, followed by $\mathsf{sk}_i := t^0_{\mathsf{A}} + t^0_{\mathsf{B}i}$. Upon finding the query index i such that $\mathsf{sk}_i \cdot G = X$, let $x := \mathsf{sk}_i$ and the problem is solved.

3) The third case:

$$H(m) \cdot \frac{e^1}{k_{\mathsf{B}}} + r_x \cdot \frac{e^{2\mathrm{a}} + t^0_{\mathsf{B}} \cdot e^{2\mathrm{b}}}{k_{\mathsf{B}}} = 0 \quad \wedge \quad e^{2\mathrm{b}} = 0$$

In this case, we give a reduction to the Computational Diffie-Hellman Problem. Recall that

$$\begin{aligned} t^1_{\mathsf{A}} + \theta &= \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^1}{k_{\mathsf{B}}} + \delta^1 \\ t^2_{\mathsf{A}} + t^2_{\mathsf{B}} &= \frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^{2\mathrm{a}} + t^0_{\mathsf{B}} \cdot e^{2\mathrm{b}}}{k_{\mathsf{B}}} + \delta^2 \end{aligned}$$

and that Bob computes $\Gamma^2 := t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk}$. Substituting the previous pair of equations into this yields

$$\begin{aligned} \Gamma^2 = &\left(\frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^{2\mathrm{a}} + e^{2\mathrm{b}} \cdot t^0_{\mathsf{B}}}{k_{\mathsf{B}}} + \delta^2 - t^2_{\mathsf{A}} \right) \cdot G \\ &- \left(\frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e^1}{k_{\mathsf{B}}} + \delta^1 - t^1_{\mathsf{A}} \right) \cdot \mathsf{pk} \end{aligned}$$

Next, according to the premise of this case, $e^{2\mathrm{b}} = 0$; thus, simplifying, we have

$$\Gamma^2 = \left(\delta^2 - t^2_{\mathsf{A}}\right) \cdot G + \left(t^1_{\mathsf{A}} - \delta^1\right) \cdot \mathsf{pk} + \frac{e^{2\mathrm{a}} - e^1 \cdot \mathsf{sk}}{k_{\mathsf{B}}} \cdot G$$

Also by the premise of this case, $e^1 = \left(-r_x \cdot e^{2\mathrm{a}}\right) / H(m)$. Given that e^1 and $e^{2\mathrm{a}}$ must be nonzero, it follows that

$$e^{2\mathrm{a}} - e^1 \cdot \mathsf{sk} = 0 \iff r_x / H(m) = 1$$

Recall that Alice is able to sample multiple values of R', from which r_x is derived via the random oracle. For each value she tries, she achieves $r_x = H(m)$ with probability $2^{-\kappa}$, and because she is polynomially bounded, she achieves $r_x = H(m)$ with probability $\mathrm{poly}(\kappa)/2^\kappa$ overall. Thus $e^{2\mathrm{a}} - e^1 \cdot \mathsf{sk} \neq 0$ with overwhelming probability, and we can rearrange such that

$$G / k_{\mathsf{B}} = \frac{\Gamma^2 + \left(\delta^1 - t^1_{\mathsf{A}}\right) \cdot \mathsf{pk} + \left(t^2_{\mathsf{A}} - \delta^{2\mathrm{a}}\right) \cdot G}{e^{2\mathrm{a}} - e^1 \cdot \mathsf{sk}}$$

Thus if Alice can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ when this case holds, then she can be used to solve the Computational Diffie-Hellman Problem with a polynomial reduction in probability in the following way: given $X = x \cdot G$, for which x is unknown, run hybrid experiment $\mathcal{H}_4$ with $D_{\mathsf{B}} := X$. Unlike the previous two cases, we have no mechanism by which to confirm our results are correct, and so we cannot identify Γ^1 reliably. Instead, choose values for Γ^1 and Γ^2 at random from the set $\mathbf{Q}$ of Alice's random oracle queries and then apply the above equation. If Alice succeeds with probability ε, then this algorithm recovers G/x with probability $\varepsilon/|\mathbf{Q}|^2$, which is bounded by $\varepsilon/\mathrm{poly}(\kappa)$ given that Alice runs in polynomial time. Given that G/x can be calculated with probability $\varepsilon/\mathrm{poly}(\kappa)$, by Corollary F.2.1 there exists an algorithm to solve the Computational Diffie-Hellman problem with probability $\varepsilon^6/\mathrm{poly}(\kappa)$.

These three cases are comprehensive; thus, if Alice can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ with non-negligible probability, then there must exist a probabilistic polynomial time algorithm that can solve either the Discrete Logarithm Problem or the Computational Diffie-Hellman Problem with non-negligible probability. Consequently, $\mathcal{H}_4$ is computationally indistinguishable from $\mathcal{H}_3$ under the Computational Diffie-Hellman Assumption, and if in future hybrids there is no abort, then e^1, $e^{2\mathrm{a}}$, and $e^{2\mathrm{b}}$ must all be equal to zero.

Hybrid $\mathcal{H}_5$. This hybrid implements Step 6 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,A}}_{n\mathsf{P\text{-}ECDSA}}$. This hybrid differs from the last in that Γ^1 is given by

$$\Gamma^1 := G + \phi \cdot k_{\mathsf{A}} \cdot G - t^1_{\mathsf{A}} \cdot R + \delta^1 \cdot R$$

rather than $\Gamma^1 := t^1_{\mathsf{B}} \cdot R$ as in $\mathcal{H}_4$. Note that Alice always queries Γ^1, and because e^1, $e^{2\mathrm{a}}$, and $e^{2\mathrm{b}}$ are zero when no abort occurs, as established in $\mathcal{H}_4$, the two derivations of Γ^1 are equivalent

$$\begin{aligned} \Gamma^1 &= t^1_{\mathsf{B}} \cdot R \\ &= \left(\frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{\phi}{k_{\mathsf{B}}} + \delta^1 - t^1_{\mathsf{A}} \right) \cdot R \\ &= G + \phi \cdot k_{\mathsf{A}} \cdot G + \delta^1 \cdot R - t^1_{\mathsf{A}} \cdot R \end{aligned}$$

Consequently, $\mathcal{H}_5$ is distributed identically to $\mathcal{H}_4$.

Hybrid $\mathcal{H}_6$. This hybrid implements Step 7 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,A}}_{n\mathsf{P\text{-}ECDSA}}$, which implies two major differences relative to $\mathcal{H}_5$. First, Γ^2 is computed as

$$\Gamma^2 \coloneqq \left(t^1_{\mathsf{A}} - \delta^1\right) \cdot \mathsf{pk} - \left(t^2_{\mathsf{A}} - \delta^2\right) \cdot G$$

rather than being computed as

$$\Gamma^2 \coloneqq \left(t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk}\right)$$

as in $\mathcal{H}_5$. These two derivations are equivalent; observe that

$$\begin{aligned}\Gamma^2 &= \left(t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk}\right)\\ &= \left(\begin{array}{l}\left(\dfrac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \delta^2 - t^2_{\mathsf{A}}\right) \cdot G\\ -\left(\dfrac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \delta^1 - t^1_{\mathsf{A}}\right) \cdot \mathsf{pk}\end{array}\right)\\ &= \left(t^1_{\mathsf{A}} - \delta_1\right) \cdot \mathsf{pk} - \left(t^2_{\mathsf{A}} - \delta^2\right) \cdot G\end{aligned}$$

Second, $\mathcal{H}_6$ aborts if

$$\mathsf{sig}_{\mathsf{A}} \neq H(m) \cdot t^1_{\mathsf{A}} + r_x \cdot t^2_{\mathsf{A}} - H(m) \cdot \delta^1 - r_x \cdot \delta^2$$

rather than aborting if $\mathsf{Verify}(\mathsf{sig}_{\mathsf{A}} + \mathsf{sig}_{\mathsf{B}}) \neq 1$ as in $\mathcal{H}_5$. Observe, however, that there is only one value of sig which verifies as a signature once R and pk are fixed, and that

$$\begin{aligned}\mathsf{sig} &= \mathsf{sig}_{\mathsf{A}} + \mathsf{sig}_{\mathsf{B}}\\ &= \mathsf{sig}_{\mathsf{A}} + H(m) \cdot \theta + r_x \cdot t^2_{\mathsf{B}}\\ &= \mathsf{sig}_{\mathsf{A}} + H(m) \cdot \left(\frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \delta^1 - t^1_{\mathsf{A}}\right)\\ &\quad + r_x \cdot \left(\frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \delta^2 - t^2_{\mathsf{A}}\right)\\ &= \mathsf{sig}_{\mathsf{A}} + \frac{H(m) + \mathsf{sk} \cdot r_x}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}}\\ &\quad - H(m) \cdot t^1_{\mathsf{A}} - r_x \cdot t^2_{\mathsf{A}} + H(m) \cdot \delta^1 + r_x \cdot \delta^2\\ &= \mathsf{sig} + \mathsf{sig}_{\mathsf{A}}\\ &\quad - H(m) \cdot t^1_{\mathsf{A}} - r_x \cdot t^2_{\mathsf{A}} + H(m) \cdot \delta^1 + r_x \cdot \delta^2\end{aligned}$$

Thus $\mathsf{Verify}(\mathsf{sig}_{\mathsf{A}} + \mathsf{sig}_{\mathsf{B}}) = 1$ if and only if

$$\mathsf{sig}_{\mathsf{A}} = H(m) \cdot t^1_{\mathsf{A}} + r_x \cdot t^2_{\mathsf{A}} - H(m) \cdot \delta^1 - r_x \cdot \delta^2$$

and $\mathcal{H}_6$ is distributed identically to $\mathcal{H}_5$.

Hybrid $\mathcal{H}_7$. In this, the last hybrid for instances of the $\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}$ where $\mathcal{P}^*_i$ plays the role of Alice, Steps 1 and 2 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,A}}_{n\mathsf{P\text{-}ECDSA}}$ are added, which implies that $\mathcal{S}^{\mathsf{2P\text{-}Sign,A}}_{n\mathsf{P\text{-}ECDSA}}$ is implemented completely, and no elements of Bob's view are required to run the experiment. These changes are syntactic, and so the distribution of $\mathcal{H}_7$ is identical to that of $\mathcal{H}_6$.

C. Simulating Against Bob

Having dealt in Appendix F-B with the view of $\mathcal{P}^*_i$ in its evaluations of $\pi^{\mathsf{2P\text{-}Sign}}_{n\mathsf{P\text{-}ECDSA}}$ with parties $\mathcal{P}_j$ where $j > i$, we now consider its view where $j < i$, and show via the series of hybrids in this section that this view is indistinguishable from that generated by the simulator $\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{SampledECDSA}}$. For convenience we refer to $\mathcal{P}^*_i$ as Bob in these hybrids.

Unlike Alice, Bob cannot cheat by inducing additive offsets into the outputs of the multiplications; he can only use inconsistent inputs. On the other hand, because he receives the output of the protocol, which is a (publicly verifiable) signature, he is able in some cases to guess values and check on his own whether his guesses are correct. Finally, note that because Bob receives the final message in our protocol, the failure condition for the protocol corresponds to the case when he cannot decrypt this message or produce a valid output. For convenience, we refer to this condition as an abort in the following section.

Simulator 8. 2-of-n Signing against Bob ($\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$):

This simulator interposes between a malicous Bob and the corresponding ideal functionality $\mathcal{F}_{\mathsf{SampledECDSA}}$. It receives as input the public key pk, the message m, the signature id $\mathsf{id}^{\mathsf{sig}}$, and Bob's share of the secret key t^0_{B}. It is parameterized by the statistical security parameter s and the group $\mathbb{G}$ of order q generated by G, with $\kappa = |q|$. It plays the roles of the functionalities $\mathcal{F}_{\mathsf{Mul}}$ and $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$ in their interactions with Bob, and it can both observe Bob's queries to the random oracle H, and program the oracle's responses.

Multiplication and Instance Key Agreement:

1) Upon receiving D_{B} from Bob, begin a new signature with Alice as the counterparty by sending $(\texttt{newsig}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{A})$ to $\mathcal{F}_{\mathsf{SampledECDSA}}$.
2) Interact with Bob on behalf of $\mathcal{F}_{\mathsf{Mul}}$, and in doing so receive his private multiplication inputs β^1, $\beta^{2\mathrm{a}}$, and $\beta^{2\mathrm{b}}$ and his corresponding outputs t^1_{B}, $t^{2\mathrm{a}}_{\mathsf{B}}$, and $t^{2\mathrm{b}}_{\mathsf{B}}$. Note that Bob's inputs to the multiplications may be inconsistent.
3) If
$$\left(G/\beta^1 = D_{\mathsf{B}}\right)$$
then let $k_{\mathsf{B}} \coloneqq 1/\beta^1$ and continue. Otherwise, abort by skipping to step 8.
4) If
$$\left(\beta^1 = \beta^{2\mathrm{a}}\right) \wedge \left(\beta^1 \cdot t^0_{\mathsf{B}} = \beta^{2\mathrm{b}}\right)$$
then continue. Otherwise, abort by skipping to step 8.
5) Send $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}})$ to $\mathcal{F}_{\mathsf{SampledECDSA}}$ and receive $(\texttt{signature}, R, k^{\Delta}, \sigma)$ in response.

Consistency Check, Signature, and Verification:

6) Compute $R' \coloneqq R - k^{\Delta} \cdot G$ and send R' to Bob. Program the random oracle H such that when it receives a query on the value R', it replies with the value $k^{\Delta}/k_{\mathsf{B}}$.
7) Interact with Bob on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$, and upon receiving $(\texttt{prove}, R, \hat{D_{\mathsf{B}}})$, reply with $(\texttt{accept}, \mathsf{A})$ if $R = H(R') \cdot \hat{D_{\mathsf{B}}} + R'$ or $(\texttt{fail}, \mathsf{A})$ otherwise.
8) Compute $\Gamma^1 \coloneqq t^1_{\mathsf{B}} \cdot R$ as Bob would if he were honest. Sample $\phi \leftarrow \mathbb{Z}_q$ uniformly and compute $\eta^{\phi} \coloneqq H(\Gamma^1) + \phi$. Send η^{ϕ} to Bob.

9) Parse $\sigma = (\mathsf{sig}, r_x)$, and then compute

$$\begin{aligned}\theta &:= t^1_{\mathsf{B}} - \phi / k_{\mathsf{B}} \\ \Gamma^2 &:= (t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk}) \\ \mathsf{sig}_{\mathsf{B}} &:= H(m) \cdot \theta + r_x \cdot t^2_{\mathsf{B}} \\ \mathsf{sig}_{\mathsf{A}} &:= \mathsf{sig} - \mathsf{sig}_{\mathsf{B}} \\ \eta^{\mathsf{sig}} &:= H(\Gamma^2) + \mathsf{sig}_{\mathsf{A}}\end{aligned}$$

Send η^{sig} to Bob and halt *without* proceeding to step 8.

Abort:

8) Send $R' \leftarrow \mathbb{G}$, $\eta^{\phi} \leftarrow \mathbb{Z}_q$, and $\eta^{\mathsf{sig}} \leftarrow \mathbb{Z}_q$ to Bob. Interact with Bob on behalf of $\mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}$, and upon receiving $(\texttt{prove}, R, \hat{D_{\mathsf{B}}})$, reply with $(\texttt{accept}, \mathsf{A})$ if $R = H(R') \cdot \hat{D_{\mathsf{B}}} + R'$ or $(\texttt{fail}, \mathsf{A})$ otherwise. Instruct $\mathcal{F}_{\mathsf{SampledECDSA}}$ to abort, and then halt. Note that this step is only reached when Bob has cheated.

For the next hybrid, we rely on circular secure encryption in the Random Oracle Model, as formalized in Lemma F.3.

Lemma F.3. *Let q be the order of a group $\mathbb{G}$ generated by G, whose elements are represented in $\kappa = |q|$ bits. If $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ is a random oracle and $x \leftarrow \mathbb{Z}_q$ is a private value sampled uniformly at random, then for any public constants $C^1, C^2 \in \mathbb{G}$ such that $C^2 \neq 0$, a PPT algorithm running in time $\mathrm{poly}(\kappa)$ with oracle access to H has an advantage no greater than $\mathrm{poly}(\kappa)/2^{\kappa}$ in distinguishing the distribution of $H(C^1 + x \cdot C^2) + x$ from the uniform distribution over $\mathbb{Z}_q$.*

Proof. The algorithm can distinguish a sample drawn from $H(C^1 + x \cdot C^2) + x$ from uniform only by guessing the correct value of x, querying H, and testing whether the result matches. Given that the algorithm can make at most $\mathrm{poly}(\kappa)$ queries to H and that $C^1 + x \cdot C^2$ is distributed uniformly over $\mathbb{Z}_q$, the probability that this point is queried to H is $\mathrm{poly}(\kappa)/2^{\kappa}$. □

Hybrid $\mathcal{H}_8$. This hybrid implements Steps 2 and 3 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$. Relative to $\mathcal{H}_7$, this experiment always aborts if Bob uses $\beta^1 \neq 1/k_{\mathsf{B}}$ as input to his first invocation of $\mathcal{F}_{\mathsf{Mul}}$, whereas in previous hybrids, Bob could use such an inconsistent input and nevertheless avoid an abort by guessing certain intermediate values and checking whether a valid signature is produced. We will show that he succeeds in avoiding an abort with negligible probability in $\mathcal{H}_7$.

Suppose that Bob passes $\beta^1 := 1/k_{\mathsf{B}} + e$ to his first invocation of $\mathcal{F}_{\mathsf{Mul}}$, instead of $1/k_{\mathsf{B}}$ as he is supposed to do. The output of this invocation will then take the form

$$t^1_{\mathsf{A}} + t^1_{\mathsf{B}} = \frac{\phi}{k_{\mathsf{B}}} + \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \phi \cdot e + \frac{e}{k_{\mathsf{A}}}$$

and Alice will compute the first check value as

$$\begin{aligned}\Gamma^1 &:= G + \phi \cdot k_{\mathsf{A}} \cdot G - t^1_{\mathsf{A}} \cdot R \\ &= G + \phi \cdot k_{\mathsf{A}} \cdot G \\ &\quad - \left(\frac{\phi}{k_{\mathsf{B}}} + \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \phi \cdot e + \frac{e}{k_{\mathsf{A}}} - t^1_{\mathsf{B}}\right) \cdot R \\ &= \left(t^1_{\mathsf{B}} - \phi \cdot e - \frac{e}{k_{\mathsf{A}}}\right) \cdot R\end{aligned}$$

She will then encrypt ϕ and transmit her encryption to Bob

$$\eta^{\phi} := \phi + H\left(\left(t^1_{\mathsf{B}} - \phi \cdot e - \frac{e}{k_{\mathsf{A}}}\right) \cdot R\right)$$

By Lemma F.3 and the fact that ϕ is drawn uniformly, Bob cannot distinguish η^{ϕ} from uniform (and thereby recover ϕ) with probability better than $\mathrm{poly}(\kappa)/2^{\kappa}$. Subsequently, Alice computes $\Gamma^2 := t^1_{\mathsf{A}} \cdot \mathsf{pk} - t^2_{\mathsf{A}} \cdot G$, which implies

$$\begin{aligned}\Gamma^2 &= t^1_{\mathsf{A}} \cdot \mathsf{pk} - t^2_{\mathsf{A}} \cdot G \\ &= \left(\frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e}{k_{\mathsf{A}}} + \frac{\phi}{k_{\mathsf{B}}} - t^1_{\mathsf{B}}\right) \cdot \mathsf{pk} \\ &\quad - \left(\frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} - t^2_{\mathsf{B}}\right) \cdot G \\ &= \left(\frac{e}{k_{\mathsf{A}}} + \frac{\phi}{k_{\mathsf{B}}}\right) \cdot \mathsf{pk} - \left(t^1_{\mathsf{B}} \cdot \mathsf{pk} - t^2_{\mathsf{B}} \cdot G\right)\end{aligned}$$

Because Bob can calculate ϕ with probability $\mathrm{poly}(\kappa)/2^{\kappa}$, his probability of deriving Γ^2 and correctly decrypting Alice's message η^{sig} is also bounded by $\mathrm{poly}(\kappa)/2^{\kappa}$. Thus, if Bob passes $\beta^1 := 1/k_{\mathsf{B}} + e$ to his first invocation of $\mathcal{F}_{\mathsf{Mul}}$ in $\mathcal{H}_7$, the experiment aborts with overwhelming probability, and consequently $\mathcal{H}_7$ and $\mathcal{H}_8$ are computationally indistinguishable.

Hybrid $\mathcal{H}_9$. In this hybrid experiment, Step 4 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$ is partially implemented. Specifically, if Bob uses $\beta^{2\mathrm{a}} \neq \beta^1$ as his input to the second invocation of $\mathcal{F}_{\mathsf{Mul}}$, then the values η^{ϕ} and η^{sig} are sampled uniformly. Because our real-world protocol coalesces Bob's first and second invocations of $\mathcal{F}_{\mathsf{Mul}}$ as described in Section VI-C, Bob is in fact already constrained to using $\beta^1 = \beta^{2\mathrm{a}}$. Therefore, $\mathcal{H}_9$ is distributed identically to $\mathcal{H}_8$, and if $\beta^{2\mathrm{a}} \neq 1/k_{\mathsf{B}}$, then then the values η^{ϕ} and η^{sig} are sampled uniformly.

Hybrid $\mathcal{H}_{10}$. This hybrid experiment fully implements Step 4 of $\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$. Bob's view in this hybrid differs from his view in $\mathcal{H}_9$ in that η^{ϕ} and η^{sig} are replaced with uniform values if Bob uses $\beta^{2\mathrm{b}} \neq t^0_{\mathsf{B}}/k_{\mathsf{B}}$ as his input to the third invocation of $\mathcal{F}_{\mathsf{Mul}}$. We must reason about the power that Bob has to guess intermediate values and correctly decrypt Alice's final message η^{sig} in the case that he has cheated in this way. Recall that to decrypt the final message, Bob computes

$$\mathsf{sig} = \mathsf{sig}_{\mathsf{B}} + \eta^{\mathsf{sig}} - H(\Gamma^2)$$

If he does not query the random oracle on the correct value of Γ^2 (in either this hybrid or $\mathcal{H}_9$), then he must guess its output, which he succeeds at doing with no better probability than he has of guessing a valid signature from whole cloth. Suppose, however, that he cheats by passing $\beta^{2\mathrm{b}} = t^0_{\mathsf{B}}/k_{\mathsf{B}} + e$ to the third invocation of $\mathcal{F}_{\mathsf{Mul}}$, and nevertheless queries the correct value of Γ^2. We will show that if he can achieve this with non-negligible probability, then he can be used to break the Computational Diffie-Hellman Assumption.

We have previously established that if there is no abort, then $\beta^1 = 1/k_{\mathsf{B}}$ and $\alpha^1 = 1/k_{\mathsf{A}} + \phi$, and the first invocation of $\mathcal{F}_{\mathsf{Mul}}$ yields shares to Alice and Bob such that

$$t^1_{\mathsf{A}} + t^1_{\mathsf{B}} = \frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{\phi}{k_{\mathsf{B}}}$$

Bob is thus able to calculate

$$\begin{aligned} \Gamma^1 &\coloneqq t^1_{\mathsf{B}} \cdot R \\ \phi &\coloneqq \eta^{\mathsf{sig}} - H(\Gamma^1) \\ \theta &\coloneqq t^1_{\mathsf{B}} - \phi/k_{\mathsf{B}} \end{aligned}$$

If Bob supplies $\beta^{2\mathrm{b}} = t^0_{\mathsf{B}}/k_{\mathsf{B}} + e$ as his input to the third invocation of $\mathcal{F}_{\mathsf{Mul}}$, then it will yield shares such that

$$t^{2\mathrm{b}}_{\mathsf{A}} + t^{2\mathrm{b}}_{\mathsf{B}} = \frac{t^0_{\mathsf{B}}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e}{k_{\mathsf{A}}}$$

which implies that

$$t^2_{\mathsf{A}} + t^2_{\mathsf{B}} = \frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e}{k_{\mathsf{A}}}$$

We have supposed that Bob queries the random oracle on the same value of Γ^2 that Alice uses to encrypt η^{sig}; thus

$$\begin{aligned} \Gamma^2 &= t^1_{\mathsf{A}} \cdot \mathsf{pk} - t^2_{\mathsf{A}} \cdot G \\ &= \left(\frac{1}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} - \theta\right) \cdot \mathsf{pk} - \left(\frac{\mathsf{sk}}{k_{\mathsf{A}} \cdot k_{\mathsf{B}}} + \frac{e}{k_{\mathsf{A}}} - t^2_{\mathsf{B}}\right) \cdot G \\ &= t^2_{\mathsf{B}} \cdot G - \frac{e}{k_{\mathsf{A}}} \cdot G - \theta \cdot \mathsf{pk} \end{aligned}$$

which in turn implies

$$G/k_{\mathsf{A}} = \frac{t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk} - \Gamma^2}{e}$$

Thus, given a (uniform) challenge $X = x \cdot G$ for which x is unknown, we can use Bob in the following way to compute G/x with a polynomial probability loss. First, sample a random value $h \leftarrow \mathbb{Z}_q$, and then run the hybrid experiment $\mathcal{H}_{10}$, using $R' \coloneqq k_{\mathsf{B}} \cdot X - h \cdot D_{\mathsf{B}}$. Program the random oracle such that $H(R') = h$; this implies that

$$R = H(R') \cdot D_{\mathsf{B}} + R' = x \cdot k_{\mathsf{B}} \cdot G$$

which is uniform, as it is in the real execution, due to the fact that x is uniform. Extract Bob's input $\beta^{2\mathrm{b}}$ to the third invocation of $\mathcal{F}_{\mathsf{Mul}}$, along with his input $1/k_{\mathsf{B}} = \beta^1$ to the first invocation, and t^0_{B}, his secret key share chosen during setup, and with these values calculate

$$e \coloneqq \beta^{2\mathrm{b}} - t^0_{\mathsf{B}}/k_{\mathsf{B}}$$

Finally, if Bob terminates with an output, choose one of his random oracle queries at random to be the value Γ^2, and then apply the equation above to compute

$$G/x \coloneqq \frac{t^2_{\mathsf{B}} \cdot G - \theta \cdot \mathsf{pk} - \Gamma^2}{e}$$

If Bob queries the correct value of Γ^2 with probability ε, and queries at most polynomially many values in total, then this reduction computes G/x with probability $\varepsilon/\operatorname{poly}(\kappa)$. By Corollary F.2.1, this implies that Bob can be used to solve the Computational Diffie-Hellman Problem with probability $\varepsilon^6/\operatorname{poly}(\kappa)$. Consequently $\mathcal{H}_{10}$ is computationally indistinguishable from $\mathcal{H}_9$ under the Computational Diffie-Hellman Assumption.

Hybrid $\mathcal{H}_{11}$. This hybrid implements the remaining steps in $\mathcal{S}^{\mathsf{2P\text{-}Sign,B}}_{n\mathsf{P\text{-}ECDSA}}$, to completely simulate Bob's view. The changes from the last hybrid are merely syntactic, and thus $\mathcal{H}_{11}$ and $\mathcal{H}_{10}$ are distributed identically.

The party $\mathcal{P}^*_i$'s view is entirely simulated in $\mathcal{H}_{11}$, and so

$$\mathcal{H}_{11} = \left\{\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{SampledECDSA}}, \mathcal{S}^{\mathsf{2P\text{-}Epoch},i}_{n\mathsf{P\text{-}ECDSA}}, \mathcal{Z}}\,(z)\right\}_{z \in \{0,1\}^*}$$

By this sequence of hybrids, $\mathcal{H}_{11} \overset{c}{\equiv} \mathcal{H}_0$; in other words, the view of a static adversary corrupting a single party $\mathcal{P}^*_i$ 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 $(\mathcal{F}_{\mathsf{Mul}}, \mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{ZK}}, \mathcal{F}^{R_{\mathsf{DL}}}_{\mathsf{Com\text{-}ZK}})$-hybrid Random Oracle Model, and Theorem F.1 is proved. □