Threshold ECDSA in Three Rounds*

Jack Doerner
j@ckdoerner.net
Technion, Reichman U, Brown U

Yashvanth Kondi
yash@ykondi.net
Silence Labs (Deel)

Eysa Lee
eysa_lee@brown.edu
Brown University

abhi shelat
abhi@neu.edu
Northeastern University

December 14, 2023

Abstract

We present a three-round protocol for threshold ECDSA signing with malicious security against a dishonest majority, which information-theoretically UC-realizes a standard threshold signing functionality, assuming only ideal commitment and two-party multiplication primitives. Our protocol combines an intermediate representation of ECDSA signatures that was recently introduced by Abram et al. [ANO+22] with an efficient statistical consistency check reminiscent of the ones used by the protocols of Doerner et al. [DKLs18, DKLs19]. We show that shared keys for our signing protocol can be generated using a simple commit-release-and-complain procedure, without any proofs of knowledge, and to compute the intermediate representation of each signature, we propose a two-round vectorized multiplication protocol based on oblivious transfer that outperforms all similar constructions.

*A preliminary version [DKLs24] of this work appeared in *IEEE S&P 2024*.

Contents

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is among the most common and widely deployed cryptographic tools of any kind. Since its standardization by the US National Institute of Standards and Technology (NIST) [Nat13], it has become a ubiquitous component of the internet infrastructure. This makes it a natural and essential target for threshold cryptography; that is, for signing mechanisms that *distribute* authority among a quorum of parties larger than some threshold. Indeed, NIST has recently announced an intent to standardize threshold ECDSA schemes [BP23], much as it did the original signature,which motivates the design of schemes that are concretely efficient, yet secure under conservative assumptions and simple to analyze and implement.

In a t-of-n ECDSA scheme, any t parties can jointly sign a message under the common public key, but no group of $t-1$ corrupt parties can sign an unauthorized message, even by sending malformed protocol messages to honest parties. Such schemes are already in use to facilitate defense-in-depth security for digital assets linked to ECDSA public keys [Lin21].

Threshold signing schemes are straightforward to construct for some elliptic curve signatures, such as BLS [BLS01] and Schnorr [Sch89], but ECDSA features a non-linear signing equation that is challenging to compute in a distributed fashion. ECDSA uses a basic discrete-logarithm key pair comprising a uniform $\mathsf{sk} \leftarrow \mathbb{Z}_q$ and a public $\mathsf{pk} = \mathsf{sk} \cdot G$, where $\mathcal{G} = (\mathbb{G}, G, q)$ is the description of an elliptic curve $\mathbb{G}$ of order q that is genrated by G. A signature on a message m consists of a public nonce $R = r \cdot G$ and a value of the form $s = (a + \mathsf{sk} \cdot b)/r$, where $a = \mathsf{SHA2}(m)$ and $b = r^{\mathsf{x}}$ are effectively public coefficients, r^{x} is the x-coordinate of the curve point R, and r is a uniform secret per-signature *instance key*. The computation of s forms the core challenge of distributing the signing process efficiently.

Most approaches to computing s can be analyzed by *rewriting* the equation that defines s in terms of some specific set of operations. For example, the modular inverse operation can be rewritten in terms of multiplication and addition, and then generic multiparty computation (MPC) protocols (which support multiplication and addition natively) can be used to compute the rewritten equation. Concrete efficiency improvements can be achieved by refining the machinery of the computation, but efficiency can ultimately be bottlenecked by the rewriting of the signing equation, which may impose some minimal number or depth of nonlinear operations that can be computed securely only at a significant cost in terms of bandwidth, computation, or rounds of interaction.

In this Work. We identify a specific rewriting of the signing equation and a method to compute it that together eliminate long standing bottlenecks in round complexity without incurring additional costs elsewhere, and yield arguably-minimal MPC protocols for both key generation and signing. Specifically, we begin from the venerable Bar-Ilan and Beaver [BB89] protocol for computing inverses of secret-shared values using secure multiplication; two

fused instances of this protocol with a single denominator are used to compute the two terms that define s. The intermediate secret sharing of the signature that results from this process is similar to the one used by several prior works [LN18, ANO+22, GS22a]. Unlike prior works, we make dual use of correlated random values already present in the Bar-Ilan and Beaver protocol as implicit Message Authentication Codes (MACs) to authenticate the various parts of the computation via checks evaluated over the signing curve in a manner reminiscent of the protocols of Doerner et al. [DKLs18, DKLs19]. This obviates the auxilliary verification mechanisms used by prior works, and it eliminates the need to extract discrete logarithms via proofs of knowledge. The resulting protocol has no zero-knowledge proofs of any kind during key generation or signing, and its cost is dominated by the cost of our fused double-instance of Bar-Ilan and Beaver. Finally, we propose a refinement of the secure multiplication protocols of Doerner et al. [DKLs18, DKLs19] that meets our syntactical requirements, while noticeably improving upon the efficiency of the original.

Our protocol improves upon the state of the art in three aspects:

- **Simplicity**. Our signing protocol is constructed using only idealized commitments and multiplication—specifically, *Vector Oblivious Linear Evaluation*, or VOLE, with a vector length of two—and both are invoked only once in each direction between every pair of parties when signing a message. In addition, every party must perform six elliptic curve scalar operations for each of its counterparties, as part of a statistical check to detect malicious behaviour. Similarly, our key generation protocol comprises a single commit-and-release action, followed by one round in which the parties can trigger an abort if they received an inconsistent share of the secret key. The latter condition is detectable using only two elliptic curve scalar operations per counterparty.

- **Security**. Assuming ideal versions of (i.e. black-box access to) the commitment and VOLE primitives, our key generation and signing protocols permit a straightforward information-theoretic security analysis in the Universal Composition (UC) framework with a conservative threshold signing functionality that simply computes the signature internally and outputs it when a quorum of parties agrees to sign. Our proposed VOLE instantiation is secure in the random oracle model assuming oblivious transfer (OT), and so concrete instantiations of our protocol (i.e. instantiations wherein all ideal primitives are realized) can be founded upon any single assumption that implies OT, such as the Diffie-Hellman assumption over the signing curve.

- **Efficiency**. Assuming a two-round VOLE such as the one we propose, the components of our signing protocol can be arranged into three rounds without breaking any abstractions or performing any heuristic optimizations. This is one round fewer than the best-known protocols from specific assumptions [CGG+20, CCL+20], and two rounds fewer than the best-known protocol from general assumptions (i.e. OT) [HLNR23]. It also brings the round complexity of threshold ECDSA to par with threshold Schnorr [Lin22]

for the first time.[1] Under pipelining and in the honest-majority setting the round complexity of our protocol can be further reduced to two with no compromise to security, and in the two-party context under pipelining only a single message in each direction is necessary. In terms of computation and bandwidth, our protocol incurs minimal overhead relative to the underlying VOLE, and we demonstrate via concrete benchmarks that it is the fastest threshold ECDSA protocol to date in scenarios ranging from only a few parties in a single location, to hundreds of parties participating globally.

1.1 A Brief History of Threshold ECDSA

In order to justify the choices we made in constructing our protocol and contextualize our claim of simplicity, we give a qualitative account of the various approaches to threshold ECDSA that have developed over time. We focus on techniques and protocol structure, rather than security models, assumptions, or comparisons of efficiency. In each case, we describe a *rewriting* of the ECDSA signing equation into a specific sequence of secure operations, summarize the protocol machinery used to compute the operations in the rewritten equation, and discuss any authentication or proof mechanism necessary to bind the various operations together and prevent malicious behavior.

Honest Majority. Shortly after the original Digital Signature Algorithm (DSA) was standardized, Langford [Lan95] devised a $\sqrt{n}$-of-n protocol to distribute its computation, and shortly thereafter Gennaro et al. generalized it to any t-of-n such that $t > n/2$ [GJKR96]. Though these works came before ECDSA, they easily extend to it. Both works rewrote the signing equation such that $s = (a + \mathsf{sk} \cdot b) \cdot r$ and $R = r^{-1} \cdot G$, and performed their computations over Shamir-sharings of the secrets sk and r. When $n \geq 2t+1$, secure multiplication of Shamir-shared secrets is is easy, and if the output need not be multiplied again, then degree-reduction is unnecessary. Gennaro et al. were the first to propose computing r^{-1} via the Bar-Ilan Beaver technique [BB89]. In their protocol, uniform sharings of r and ϕ are sampled, their product $r \cdot \phi$ is publicly revealed, as is $\phi \cdot G$, and then $R = (r \cdot \phi)^{-1} \cdot \phi \cdot G = r^{-1} \cdot G$ can be computed publicly.

These first protocols were secure in the semi-honest honest majority setting, but honest majorities also permit simple and clean techniques for achieving malicious security. Gennaro et al. [GJKR96] and Cerecedo et al. [CMI93] used *verifiable secret sharing* (VSS) techniques to build *robust* protocols when $n \geq 3t + 1$, and Damgård et al. [DJN+20] constructed a simple protocol that is secure with abort against malicious adversaries when $n \geq 2t+1$. More recently, Groth and Shoup [GS22a] developed the first distributed ECDSA protocol that achieves guaranteed output delivery in the *asynchronous* setting when $n \geq 3t+1$,

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

which is optimal. Our focus in this work is the dishonest-majority setting, and so we dwell no further on the details of these protocols.

Dishonest Majority. MacKenzie and Reiter [MR01] constructed the first two-party (and thus dishonest-majority) protocol for ECDSA signing. They expressed the ECDSA signing equation as $s = (a/r) + (\mathsf{sk} \cdot b/r)$, and then specified that the parties sample *multiplicative* shares of sk and r, from which R can be computed using a Diffie-Hellman key exchange and multiplicative shares of the terms a/r and $\mathsf{sk} \cdot b/r$ can be computed non-interactively. The latter terms must be summed without revealing either of them. MacKenzie and Reiter do this via Paillier's *additively homomorphic encryption* (AHE) scheme [Pai99]. Fifteen years later, Gennaro et al. [GGN16] used *threshold* AHE to extend this idea to the many-party setting, and then Boneh et al. [BGG17] reduced the round count to four. Both many-party extensions rely on two severely inefficient components: threshold AHE requires an auxiliary RSA modulus of unknown factorization, which can only be sampled via an additional highly-complex protocol [HMRT12, FLOP18, CCD+20, CHI+21], and using Paillier encryption to operate over a prime-order field like the one in which the shares-to-be-combined lie requires expensive zero-knowledge range proofs to ensure ciphertext well-formedness. Further proofs are required to verify the relationships of the encrypted values to the public ones. Lindell [Lin17] eliminated these bottlenecks in the two-party setting by ensuring that one of the parties is given a homomorphic encryption of the other party's share of the signing key during key generation: this allows the same party to compute an encryption of s non-interactively, and the other party can simply check if the signature is valid upon decryption. Lindell's signing protocol requires a new ad-hoc assumption on the Paillier cryptosystem in order to achieve simulation security, but it is computationally limited only by the large-integer arithmetic required to work with Paillier encryption. Unfortunately, it is unclear how to generalize Lindell's improvements to the multiparty setting.

Doerner et al. [DKLs18] used the same form of the signing equation as MacKenzie and Reiter and the same multiplicative secret sharings. They combined the multiplicative shares using an ideal secure multiplication functionality, and then proposed an actively-secure variant of Gilboa's OT-based multiplication protocol [Gil99] to realize this functionality. Because their multiplication protocol operates natively over any prime-order field, it avoids large integer arithmetic and range proofs. Its compatibility with OT extension [IKNP03, KOS15, Roy22] techniques makes it computationally lightweight relative to Lindell's scheme, at the expense of somewhat higher bandwidth consumption. In a subsequent work, Doerner et al. [DKLs19] extended their scheme to the multiparty setting by introducing a $\log t$-round protocol to convert a t-party multiplicative sharing to an additive one. Both of these works [DKLs18, DKLs19] verified the consistency of intermediate computations against malicious behavior using a set of bespoke checks that relate secret shares to public values in the curve group $\mathbb{G}$, and show reductions to standard assump-

tions on $\mathbb{G}$ if the checks are violated without detection. In the multi-party follow-up these checks are carried out via the simple commit-and-release of shares that will sum to a known public target value if and only if all parties have behaved honestly. In the original two-party protocol the check shares are instead used to encrypt the final protocol message, such that it can only be decrypted correctly if both parties are honest; this trick among others allows the original protocol to produce a signature in two messages. Our work bespoke checks that are similar in spirit to those employed previously by Doerner et al., but ours are evaluated in a pairwise fashion, and they are statistical instead of computational.

Gennaro and Goldfeder [GG18] returned to Langford's forumulation of the ECDSA signing equation and devised a Paillier-based mechanism to compute it that differs from previous Paillier-based schemes [GGN16, BGG17] in that it does not require the secure sampling of biprimes of unknown factorization. Their protocol is eight rounds in total and achieves malicious security: the first three rounds resemble the protocol of Gennaro et al. [GJKR96], except that they use Paillier encryption to perform secure multiplication. After these three rounds, the signature cannot be revealed immediately: instead a five-round interactive protocol is used to perform a masked verification of the putative signature, ensuring that the signature is well-formed (and thus no cheating has occurred) before it is assembled.

Concurrently, Lindell and Nof [LN18] presented a different eight-round Paillier-based threshold ECDSA protocol that similarly avoided secure biprime sampling. Unlike Goldfeder and Gennaro's protocol, Lindell and Nof used a new rewriting of the ECDSA equation wherein $R = r \cdot G$ and $s = w/u$, where $w = (a + \mathsf{sk} \cdot b) \cdot \phi$, and $u = r \cdot \phi$ for some randomly sampled ϕ. This was the first appearance in the literature of the rewriting that we use in *this* work, i.e. the fused double-instance of Bar-Ilan an Beaver. Again like Gennaro and Goldfeder, Lindell and Nof proposed to use a "private but unauthenticated" computation mechanism for this rewriting, and then verify the well-formedness of the signature before revealing it. Their verification mechanism essentially repeats the signing computation in encrypted form, using a combination of ElGamal encryption and relatively-efficient zero-knowledge proofs. Since their analysis is in the UC model, these proofs must be compiled for straight-line extraction via the Fischlin transform [Fis05], which induces an overhead of roughly one order of magnitude in terms of computation and communication.

Somewhat later, Canetti et al. [CGG+20] presented a four-round signing protocol that essentially followed the same core protocol layout as Gennaro and Goldfeder [GG18], but they replaced the interactive masked signature verification to validate honest behaviour with a conceptuatlly-straightforward GMW-style [GMW87] mechanistm in which the parties prove honest execution of each step in zero-knowledge. The advantage of this alteration is an improved round count and the ability to identify cheating parties. The downside is the increased computational cost due to performing zero-knoweledge proofs over cryptographic statements. Parties must prove at every step that certain Paillier ciphertexts encrypt the discrete logarithms of public values, the results of some

affine operations, or values in some restricted range.

Lindell and Nof, Gennaro and Goldfeder, and Canetti et al. all instantiated their secure multiplication primitives using Paillier encryption, and thus they rely upon expensive range proofs to guarantee correctness. In a recent update to Lindell and Nof's work, Haitner et al. [HLNR23] replaced the original secure multiplication protocol with OT-based *weak* multiplication (which guarantees privacy but not correct outputs), and achieved a round count of five by making optimizations at the expense of breaching abstraction boundaries.

Castagnos et al. [CCL+23] started from the protocol of Canetti et al. and replaced Paillier encryption with the Castagnos-Laguillaumie (CL) encryption scheme [CL15] from class groups of imaginary quadratic order, which eliminated the zero-knowledge range proofs required by Canetti et al. and yielded a significant bandwidth improvement, but did not reduce the computational burden of the original protocol due the inherently higher cost of CL encryption, relative to Paillier.

Generic Approaches. Smart and Talibi [ST19] and Dalskov et al. [DOK+20] concurrently proposed generic MPC approaches to ECDSA signing. Both groups observed that the MAC checks of SPDZ-style [DPSZ12] protocols can be evaluated in an elliptic curve group directly. SPDZ-style protocols are typically black-box in a Beaver-triple generator, which is essentially an authenticated secure multiplication primitive, and since SPDZ-style protocols are generic, they can compute *any* rewriting of the ECDSA signing equation. The downside of such generic approaches was evident in the benchmarks that Dalskov et al. reported [DOK+20]: the overhead due generating MACs amounts to several additional rounds of interaction relative to other approaches, and a factor of two in terms of bandwidth and computation.

Abram et al. [ANO+22] studied how to construct threshold ECDSA in the Pseudorandom Correlation Generator (PCG) paradigm, using a variant of the Ring-LPN assumption. This involves first defining a multiparty correlation that can be derandomized into an ECDSA signature, and then constructing cryptographic machinery to derive many instances of the correlation non-interactively after a one-time setup. The correlation they defined comprises (ϕ, r, u, v) such that $u = \phi \cdot r$ and $v = \phi \cdot \mathsf{sk}$. They refer to it as an "ECDSA Tuple." Assuming a linear secret sharing scheme, this correlation can be assembled with the message into an ECDSA signature in one round by locally computing shares of $w = a \cdot \phi + b \cdot v$ and then publishing shares of both w and u, and publicly computing $s = w/u$. This correlation essentially distills Lindell and Nof's [LN18] rewriting of the ECDSA equation into a clean, succinct format, suitable for many different kinds of secure computation machinery. Abram et al. themselves took a generic approach: they augmented the correlation with BeDOZa-style MACs [BDOZ11], which can be checked in an elliptic curve group much as SPDZ-style MACs can be, and used these MACs to authenticate the honest generation and assembly of the correlation.

1.2 Our Approach and Contributions

Just as we have done with prior works, we can break down our protocol into a rewriting of the ECDSA equation, a mechanism for securely computing the operations in the rewritten equation, and an approach to hardening the secure computation against malicious adversaries.

Rewriting ECDSA. In this work, we propose a protocol that securely computes $R = r \cdot G$ and $w = (a + \mathsf{sk} \cdot b) \cdot \phi$ and $u = r \cdot \phi$, where $a = \mathsf{SHA2}(m)$ is a public coefficient, $b = r^{\times}$ is the x-coordinate of R, and r is a uniform secret. Once R, w, and u are known to the signing parties, they can information-theoretically construct an ECDSA signature by locally calculating $s = w/u$. This is the same formulation of the signing equation originally introduced by Lindell and Nof [LN18] and later explicitly construed as a random correlation by Abram et al. [ANO+22]. Unlike Lindell and Nof, we leverage the fact that under this formulation (as opposed to others), the three nonlinear relations defining R, w, and u can be securely computed in parallel. This is critical for achieving a three-round protocol. Unlike Abram et al. [ANO+22], we compute the correlation exactly as we have written it, rather than extending it with explicit BeDOZa-style MACs.

Computing the ECDSA Correlation Securely. We propose a protocol that leverages the structure of the correlation itself to achieve security against malicious adversaries, rather than relying upon zero-knowledge proofs or explicit MACs on computed values. Specifically, we propose to use a pairwise *consistency check* that depends only upon the inputs and outputs of the operations in our rewritten signing equation. This frees us to model the operations of the signing equation as ideal objects and to instantiate them modularly. Among prior works, only Doerner et al. [DKLs18, DKLs19] use a similar approach, but our consistency checks are pairwise and statistical, whereas theirs were global and computational, and the simulation strategy used in their proof requires the checks to be evaluated over the course of multiple rounds, whereas ours can be performed simultaneously with the computation of the correlation.

As we have said, our protocol securely computes $R = r \cdot G$ and $w = (a + \mathsf{sk} \cdot b) \cdot \phi$ and $u = r \cdot \phi$, where $a = \mathsf{SHA2}(m)$ is a public coefficient, $b = r^{\times}$ is the x-coordinate of R, and r is a uniform secret. The computation of secret shares of w can be performed locally by the parties given shares ϕ and $v = \mathsf{sk} \cdot \phi$. Assuming that shares of the two products v and u are computed *ideally* (i.e. in each case it is guaranteed that nothing is leaked in the course of computing the product, and that the outputs are really shares of the product of the shared inputs), there are only a few avenues to cheat:

1. A corrupt party could bias the sampling of r. This can be prevented by using a standard commit-and-release sampling mechanism: the parties sample shares of r, commit to corresponding shares of R (which collectively fix r), and then decommit the latter shares. This requires two rounds. So long as r

is otherwise information-theoretically hidden until after the commitment is complete, this precludes any bias on the part of the adversary.

2. A corrupt party could use inconsistent values of ϕ in the computations that produce v and u. This can be prevented by using a *vectorized* multiplication primitive to compute both values at once, given a single value of ϕ. Specifically, we use *vector oblivious linear evaluation* (VOLE), which is evaluated pairwise. Given any ordinary two-party two-message OLE (i.e. multiplication) protocol in which the party who speaks first supplies a share of ϕ, the party who speaks second can vectorize their input simply by reusing the first message for multiple responses. This essentially fuses the two multiplications.

3. In the computations that produce v and u, a corrupt party could use values of sk and r that are not actually the respective discrete logarithms of pk and R. To mitigate this form of attack, we devise an extremely simple consistency check mechanism hinging on the observation that if the shares of ϕ are interpreted as MAC keys, then the parties are *already* in possession of BeDOZa-style MACs on each other's shares of sk and r. Furthermore, these MACs can be checked in the elliptic curve group against the publicly known values of pk and R, and if the party that supplies a share of ϕ speaks first in the multiplication protocol, and the protocol requires two messages, then the check can be performed *simultaneously* and without additional messages.

   Let us be more specific: ϕ_i is party $\mathcal{P}_i$'s share of ϕ, r_j is $\mathcal{P}_j$'s share of r, and $R_j = r_j \cdot G$ is known to both parties. The two parties use a two-message multiplication protocol that privately outputs c to $\mathcal{P}_j$ and d to $\mathcal{P}_i$ such that $c + d = r_j \cdot \phi_i$. If the multiplication protocol involves two messages and $\mathcal{P}_i$ speaks first, then $\mathcal{P}_j$ must learn c after receiving the first message from $\mathcal{P}_i$. To ensure consistency, we specify that $\mathcal{P}_j$ transmits $\Gamma = c \cdot G$ to $\mathcal{P}_i$ along with the second message of the multiplication protocol, and then $\mathcal{P}_i$ checks that $\Gamma + d \cdot G = \phi_i \cdot R_j$. The same check can be performed with respect to $\mathsf{pk}_j = \mathsf{sk}_j \cdot G$.

   This statistical check is cheap, overwhelmingly sound, and extremely simple to simulate. Since Γ can be computed as a function of R_j and the secrets of $\mathcal{P}_i$, simulating this value toward $\mathcal{P}_i$ without knowledge of r_j is trivial. On the other hand, fixing R_j, ϕ_i, and d fixes exactly one value of Γ that will cause the consistency check to pass. Since ϕ_i and d are uniform and information-theoretically hidden from $\mathcal{P}_j$, the correct value of Γ can be guessed with probability at most $1/q$ if $r_j \cdot G \neq R_j$. Finally, even if the check is passed—that is, if R_j and a the correct value of Γ are known to $\mathcal{P}_j$—the remaining degree of freedom defining the relationship between ϕ_i and d ensures that ϕ_i is information-theoretically hidden from $\mathcal{P}_j$, and thus it remains safe to use ϕ_i in constructing the signature even though it is also used in this check.

4. A corrupt party could send an incorrect share of v or u, after they are computed. Since fixing m, R, and pk fixes the corresponding ECDSA signature exactly, a cheat of this kind can be detected perfectly by verifying the signature that is assembled.

We have discussed a two-round commit-and-release mechanism to compute R and a two-round fused multiplication protocol with a consistency check that requires (shares of) R to be known after the second round. These two operations can be performed concurrently. Afterward, only one more round is needed to assemble the signature from the correlation, and the final check is performed locally. Thus our protocol requires three rounds, in total. Our final theorem statement is:

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

where n is the number of parties in total, t is the threshold of parties required for a signature to be produced, and $\mathcal{G}$ is the description of an elliptic curve. In addition to the commitment functionality $\mathcal{F}_{\mathsf{Com}}$ and the randomized VOLE functionality $\mathcal{F}_{\mathsf{RVOLE}}$,[2] our protocol uses $\mathcal{F}_{\mathsf{Zero}}$ to generate secret-sharings of zero, and we abstract the key generation process behind $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$.

Machinery for Multiplication. The protocol we have just described requires a two-message vector OLE protocol in which the first party to speak supplies one value, and the second supplies two. Doerner et al. [DKLs18] gave a two-round multiplication protocol based upon *oblivious transfer* (OT), and in a follow-up work they gave a three-round variant with reduced bandwidth [DKLs19]. In this work, we refine their techniques and propose a new VOLE protocol[2] that has lower concrete bandwidth costs than *either* of their protocols, only two rounds, and no new assumptions or primitives. In other words, our new protocol is the best of both worlds. Specifically, if $\mathcal{F}_{\mathsf{EOTE}}$ is an endemic OT-extension functionality,[3] which can be realized efficiently from many public-key assumptions using well-known techniques, we prove:

Theorem 1.2 (Informal Random VOLE Security Theorem)**.** *In the* $\mathcal{F}_{\mathsf{EOTE}}$*-hybrid non-programmable global random oracle model,* $\pi_{\mathsf{RVOLE}}(q, \ell)$ *UC-realizes* $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$ *against a PPT malicious adversary that statically corrupts no more than one party.*

Since $\mathcal{F}_{\mathsf{EOTE}}$ can be instantiated efficiently assuming only the decisional or computational Diffie-Hellman assumption over the signing curve [MR19, CSW20], it is possible to implement our protocol entirely from *native* assumptions, much as Doerner et al. did [DKLs18, DKLs19]. We believe this VOLE protocol to inhabit a practically-advantageous position in the spectrum of bandwidth/computation tradeoffs, but we stress that our protocol can use *any* VOLE protocol that realizes a suitable functionality, and in particular, if bandwidth

[2] Technically, we only require and only propose a *random* VOLE protocol, but such a protocol can be trivially lifted to the standard notion of VOLE.

[3] We introduce this functionality and the rationale behind it in section 5. For now it can be thought of as as performing batches of oblivious transfers, with random inputs.

savings is paramount, then a VOLE derived from additively-homomorphic encryption [CGG+20, CCL+23] can be substituted.

Zero Zero-Knowledge Required. Because our signing protocol uses *ideal* multiplication, and our consistency check ensures that the inputs to the multiplication functionality are the discrete logarithms of R_i and pk_i for each party $\mathcal{P}_i$, the multiplication functionality can be used to extract the adversary's secrets. Viewed in this way, the multiplication and consistency check together form a designated-verifier zero-knowledge proof of knowledge. Since the adversary's secrets are required only to simulate the *last* message in the signing protocol, after the extraction has occurred, no additional proofs of knowledge are necessary, *even during key generation*. This allows us to introduce a *relaxed* key generation functionality and an extremely simple commit-release-and-complain protocol to realize that functionality, and it allows us to completely avoid the overhead typically incurred when proofs of knowledge are compiled for straight-line extraction via the Fischlin [Fis05] or Kondi-shelat [Ks22] transforms. This is particularly advantageous when new keys are generated almost as frequently as signatures, as is sometimes the case in blockchain contexts.

Bandwidth and Computational Efficiency. We show via closed-form analysis that when our VOLE protocol is used with our threshold ECDSA protocol, the overall bandwidth cost is significantly reduced relative all prior OT-based threshold ECDSA schemes [DKLs18, DKLs19, DOK+20]. In terms of bandwidth, our combined protocol is competitive with techniques based on *weak* multiplication [HMRT22, HLNR23], which require more rounds. In terms of concretely demonstrated performance (i.e. minimal wall-clock time in practical benchmarks), the protocols of Doerner et al. [DKLs18, DKLs19] have heretofore remained the state of the art due to the fact that competing approaches based upon Paillier encryption [Pai99, CGG+20] or class groups [CL15, CCL+23] have excessive computational costs. We show empirically that the protocol in this work provides a strict improvement upon and fully subsumes the works of Doerner et al.. Moreover, the modularity and simplicity of our ECDSA signing protocol imply that its performance properties can be adjusted to mimic those of threshold ECDSA schemes based on other approaches simply by replacing the VOLE with a different instantiation.

Organization of this Paper. After our preliminaries, we introduce our Threshold ECDSA protocol in section 3 and prove it secure in section 4. In section 5 we give our random VOLE construction, and we prove it secure in section 6. In section 7 we give our key generation protocol. In section 8 we give a closed-form cost analysis of all of our prototocols, and in section **??** we discuss a proof-of-concept implementation and report benchmark results in a number of settings. Finally, we give a brief account of a significantly simplified two-round honest-majority protocol in section 9.

2 Preliminaries

2.1 Notation

We use $=$ for equality, $:=$ for right-to-left assignment, $=:$ for left-to-right assignment, and $\leftarrow$ for right-to-left sampling from a distribution. Single-letter variables are set in *italic* font, function names are set in sans-serif font, and string literals are set in `slab-serif` font. We use $\mathbb{X}$ for an unspecified domain, $\mathbb{G}$ for a group, $\mathbb{Z}$ for the integers, and $\mathbb{N}$ for the natural numbers. We use λ_{c} and λ_{s} to denote the computational and statistical security parameters, respectively, and κ is the number of bits required to represent an element of the order field of an elliptic curve.[4]

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

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

2.2 Security and Communication Model

We consider a malicious PPT adversary who can statically corrupt up to $t-1$ parties. All of our proofs are expressed in the Universal Composition framework [Can01]. Our techniques do not rely on any specific properties of the framework. We assume that all of the parties in any protocol are fully connected via authenticated channels, and that the network is asynchronous. We do not assume a broadcast channel, and we do not guarantee output or termination.

2.3 The ECDSA Signature Scheme

All algorithms in the ECDSA signature scheme are parameterized by $\mathcal{G} = (\mathbb{G}, G, q)$, which is the description of an elliptic curve group $\mathbb{G}$ of order q that is generated by G. Here $\kappa = |q|$. At a minimum, security requires a curve-sampling algorithm $\mathcal{G} \leftarrow \mathsf{GrpGen}(1^{\lambda_{\mathsf{c}}})$ against which the discrete logarithm assumption must hold.[5] In practice, the group description is fixed and standardized.

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

[5] This is necessary, but not known to be sufficient; as of writing ECDSA cannot be proven secure under any standard assumption.

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

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

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

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

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

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

3 t-Party Three-Round Threshold ECDSA

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

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

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

corrupt, then the adversary $\mathcal{S}$ may instruct the functionality to abort selectively during the setup phase *only*. $\mathcal{S}$ may also instruct the functionality to fail during the signing phase if any party indexed by $\mathbf{P}$ is corrupt, but in this case the functionality does *not* halt, and further signatures may be attempted.

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

1. Sample the joint secret and public keys, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.
3. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly to $\mathcal{S}$.
4. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ from $\mathcal{S}$, send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{P}_i$ and store $(\texttt{pk-delievered}, \mathsf{sid}, i)$ in memory. On receiving $(\texttt{abort}, \mathsf{sid}, i)$, send $(\texttt{abort}, \mathsf{sid})$ to $\mathcal{P}_i$, and do not interact with $\mathcal{P}_i$ any further in this session.

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

5. If there is any pair of signers $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $\mathsf{SHA2}(m_i) \neq \mathsf{SHA2}(m_j)$, then for every $i \in \mathbf{P}$, then send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$.
6. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.
7. If a corrupt party is indexed by $\mathbf{P}$, and $\mathcal{S}$ sends $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ such that $i \in \mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ and ignore any future $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ or $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$ message.
8. If no corrupt parties are indexed by $\mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P}$.
9. Once every signing party has received an output, ignore all future messages with this sigid value.

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

3.1 Building Blocks

Here we define simpler functionalities from which our protocol will be constructed. All are standard and can be realized via standard techniques. We also give notes on realization strategies and performance.

We begin with a functionality that samples Shamir sharings of keys for discrete-log cryptosystems (e.g. ECDSA, the Schnorr signature scheme, the ElGamal encryption scheme, the BBS+ signature scheme, etc). We refer to this functionality as the *relaxed* key generation functionality, because it does not explicitly sample a secret key, and it may not even have enough information interally to compute the secret key, depending on the values of t and n. However, it *always* denies the adversary the ability to compute the secret key, assuming that the discrete logarithm problem is hard. In section 7 we discuss this design decision and its implications, and introduce a protocol to realize our functionality.

Functionality 3.2. $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$**: Relaxed DLog Keygen**

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

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

1. Receive $(\texttt{adv-poly}, \mathsf{sid}, \{\check{p}(i)\}_{i \in [n] \setminus \mathbf{P}^*}, \{\check{P}(j)\}_{j \in \mathbf{P}^*})$ from $\mathcal{S}$. Let $\check{P}(i) := \check{p}(i) \cdot G$ for $i \in [n] \setminus \mathbf{P}^*$, and abort if $\check{P}$ is not a degree-$(t-1)$ polynomial over $\mathbb{G}$.
2. Sample a degree-$(t-1)$ polynomial $\hat{p}$ uniformly over $\mathbb{Z}_q$. Let $\hat{P}(k) := \hat{p}(k) \cdot G$ and $P(k) := \check{P}(k) + \hat{P}(k)$ for all $k \in [n]$, and let $p(i) := \check{p}(i) + \hat{p}(i)$ for $i \in [n] \setminus \mathbf{P}^*$. Note that P is a polynomial of degree $t-1$ over $\mathbb{G}$. Interpolate $P(0)$.
3. Send $(\texttt{hon-poly}, \mathsf{sid}, P(0), \{\hat{P}(i)\}_{i \in [n] \setminus \mathbf{P}^*}, \{\hat{p}(j)\}_{j \in \mathbf{P}^*})$ directly to $\mathcal{S}$.
4. On receiving $(\texttt{release}, \mathsf{sid}, i)$ for $i \in [n]$ directly from $\mathcal{S}$, if $\mathcal{P}_i$ is honest, then send $(\texttt{key-pair}, \mathsf{sid}, P(0), p(i))$ to $\mathcal{P}_i$. If $\mathcal{P}_i$ is corrupt, then do nothing. If $(\texttt{abort}, \mathsf{sid}, i)$ is received instead, then send $(\texttt{abort}, \mathsf{sid})$ to $\mathcal{P}_i$.

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

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

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

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

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

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

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

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

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

Finally, we use a randomized VOLE functionality $\mathcal{F}_{\mathsf{RVOLE}}$:[6] the first party (Bob) to invoke the functionality receives a single random value; the second party (Alice) then supplies a vector of chosen values, and $\mathcal{F}_{\mathsf{RVOLE}}$ outputs to both of them secret shares of the product of the random value and each of the elements in the vector. We give a protocol to realize $\mathcal{F}_{\mathsf{RVOLE}}$ in section 5, and prove it secure in section 6.

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

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

[6] As we discuss in section 5, this is equivalent to plain VOLE under a simple information-theoretic transformation.

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

Multiplication: On receiving $(\texttt{multiply}, \mathsf{sid}, \mathbf{a})$ from Alice, where $\mathbf{a} \in \mathbb{Z}_q^\ell$, if there exists a message of the form $(\texttt{instance}, \mathsf{sid}, b)$ in memory, and if $(\texttt{complete}, \mathsf{sid})$ does not exist in memory, then:

- If Alice is corrupt, receive $(\texttt{alice-share}, \mathsf{sid}, \mathbf{c})$ from $\mathcal{S}$ and compute $\mathbf{d} := \{\mathbf{a}_i \cdot b - \mathbf{c}_i\}_{i \in [\ell]}$.
- If Bob is corrupt, send $(\texttt{alice-multiplied}, \mathsf{sid})$ to $\mathcal{S}$, wait for $(\texttt{bob-share}, \mathsf{sid}, \mathbf{d})$ in response, and compute $\mathbf{c} := \{\mathbf{a}_i \cdot b - \mathbf{d}_i\}_{i \in [\ell]}$.
- If neither party is corrupt, sample $\mathbf{c} \leftarrow \mathbb{Z}_q^\ell$ and $\mathbf{d} \leftarrow \mathbb{Z}_q^\ell$ uniformly subject to $\{\mathbf{a}_i \cdot b\}_{i \in [\ell]} = \{\mathbf{c}_i + \mathbf{d}_i\}_{i \in [\ell]}$.

and send $(\texttt{share}, \mathsf{sid}, \mathbf{c})$ to Alice, send $(\texttt{share}, \mathsf{sid}, \mathbf{d})$ to Bob, and store $(\texttt{complete}, \mathsf{sid})$ in memory.

3.2 The Basic Three-Round Protocol

In this section we give our three round signing protocol. We begin by developing some intuition, building upon the sketch of our protocol in section 1.2. Suppose that each party $\mathcal{P}_i$ knows additive shares r_i and sk_i of r and sk respectively, and samples a uniform mask ϕ_i. Suppose also that they know u_i and v_i such that

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

It is easy to see that given these correlations,

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

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

Towards ensuring consistency, our main contribution is a novel method to verify an enriched version of the correlation: each $\mathcal{P}_i$ knows $\mathbf{c}^\mathsf{u}_{i,j}$ and $\mathbf{c}^\mathsf{v}_{i,j}$ and

each $\mathcal{P}_j$ knows $\mathbf{d}^{\mathsf{u}}_{j,i}$ and $\mathbf{d}^{\mathsf{v}}_{j,i}$ such that

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

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

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

is exactly $1/q$. Thus by checking the latter equality, $\mathcal{P}_j$ can ensure that $\mathcal{P}_i$ has behaved *consistently* with overwhelming probability. A similar check allows $\mathcal{P}_j$ to ensure the consistency of pk_i and sk_i via $\mathbf{c}^{\mathsf{v}}_{i,j}$ and $\mathbf{d}^{\mathsf{v}}_{j,i}$. Finally, it is easy to compute an appropriate value u_i given knowledge of r_i, ϕ_i, $\mathbf{c}^{\mathsf{u}}_{i,*}$, and $\mathbf{d}^{\mathsf{u}}_{i,*}$, and to compute an appropriate v_i given knowledge of sk_i, ϕ_i, $\mathbf{c}^{\mathsf{v}}_{i,*}$, and $\mathbf{d}^{\mathsf{v}}_{i,*}$, which implies that signature assembly can happen as before.

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

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

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

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

Setup:

1. On receiving $(\mathtt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks

whether there exists a record of the form $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ in memory. If not, then $\mathcal{P}_i$ sends $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.

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

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

Signing:

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

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

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

6. $\mathcal{P}_i$ sends

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

   This completes the first round.

if pipelining, supply m here[b]

7. On receiving

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

   $\mathcal{P}_i$ computes

$$\mathsf{sk}_i := p(i) \cdot \mathsf{lagrange}(\mathbf{P}, i, 0) + \zeta_i$$

and sends $(\texttt{multiply}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{r_i, \mathsf{sk}_i\})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for $j \in \mathbf{P}^{\text{-}i}$, and receives $(\texttt{share}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{c}^{\mathsf{u}}_{i,j}, \mathbf{c}^{\mathsf{v}}_{i,j}\})$ for $j \in \mathbf{P}^{\text{-}i}$ in response. Then $\mathcal{P}_i$ computes

$$\begin{aligned}
\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} &:= \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G \\
\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} &:= \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G \\
\mathsf{pk}_i &:= \mathsf{sk}_i \cdot G \\
\boldsymbol{\psi}_{i,j} &:= \phi_i - \boldsymbol{\chi}_{i,j}
\end{aligned}$$

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

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

if presigning, supply m here[b]

8. On receiving

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Pipelining and Presigning

We have marked the above protocol in two places to show how it can be modified to add *pipelining* or *presigning* in order to reduce the number of rounds under certain circumstances (like several previous works [DOK+20, CGG+20, CCL+23]). In each case, the parties must supply the message m to the protocol at the indicated point, instead of at the beginning of the protocol.

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

parties are activated by the environment, and potentially before the corrupt parties are. This necessitates a revised functionality, which we present below.

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

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

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

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

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

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

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

7. If the value of m submitted is not consistent among all parties, then for every $i \in \mathbf{P}$, wait for $\mathcal{S}$ $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, i)$ and then send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_i$.

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

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

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

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

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

3.4 Comparison to DKLs19

The clearest single ancestor of our protocol is the t-of-n signing protocol of Doerner et al. [DKLs19], hereafter referred to as the 2019 DKLs protocol. Although our protocol contains some of the same fundamental ideas as that one, ours rearranges the main protocol structure and the eliminates an intermediate functionality (the so-called inverse-sampling functionality) to yield a significant improvement in the number of rounds and a completely new information-theoretic proof. Specifically, whereas the 2019 DKLs protocol requires either $6 + \log t$ or 10 rounds under the computational Diffie-Hellman assumption in the signing curve, our new protocol requires only 3 rounds (one of which is pipelineable) and is statistically secure without a random oracle. Both protocols are otherwise expressed in similar hybrid models. Our new protocol requires exactly as many secure multiplications to be performed as does the 10-round version of the 2019 protocol, whereas the $(6 + \log t)$-round version requires fewer. In spite of this fact, our improvements to the secure multiplication protocol (i.e. our random VOLE) ensure that our new protocol has a lower bandwidth cost overall, as we discuss in section 8. While the number of rounds is significantly

improved relative to the 2019 DKLs scheme, we note that the number of elliptic curve scalar operations grows with the number of signers in our new scheme, whereas in the 2019 DKLs scheme it is a constant.

The heart of the structural difference between the two protocols lies in the way they compute shares of $1/r$ and sk/r, and in the way they check the correctness of these computations. In the 2019 DKLs protocol, a distinct functionality is defined to sample R along with shares of r and $1/r$. This functionality is realized by a protocol that samples multiplicative shares of r, inverts them locally, and then uses a $O(\log t)$-long sequence of pairwise secure multiplications to compute additive shares of both r and ϕ/r, where ϕ is a uniform mask. A single commit-and-release check assures the well-formedness of the shares in the 2019 scheme, before they are unmasked. At this point shares of $1/r$ are multiplied by shares of sk, and an additional commit-and-release check establishes the correctness of this multiplication with respect to pk. The 2019 protocol's higher round count is due the fact that it performs the inversion and multiplication operations sequentially, and the fact that it performs two sequential commit-and-release checks. In contrast the protocol introduced here performs inversion, multiplication with sk, and consistency checking simultaneously.

3.5 Two-Party Two-Message ECDSA

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

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

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

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

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

We note that these modifications to the protocol are secure because they are essentially equivalent to rushing behavior, and our proof in section 4 already acounts for rushing adversaries. We illustrate the modifications in figure 1.

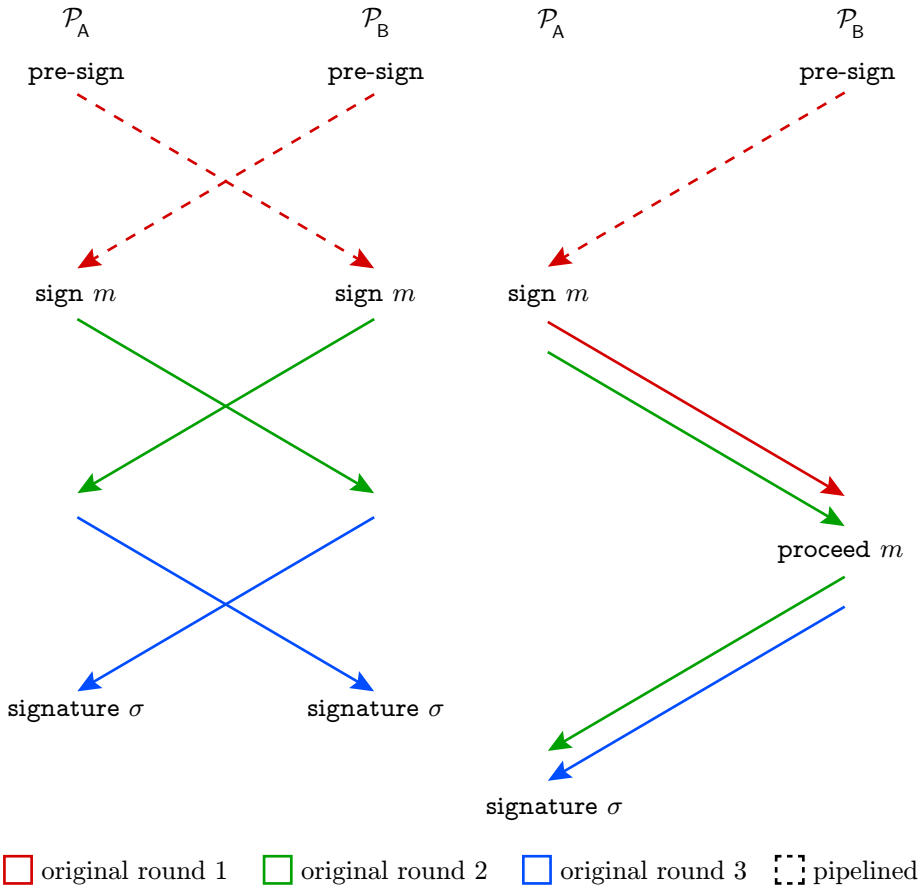

Figure 1: Two-party Message Structures Illustrated. On the left is the protocol structure, with pipelining, as described in sections 3.2 and 3.3. On the right is the protocol structure suggested for the two-party setting in this section.

The resulting protocol comprises three messages, and if the parties pipeline the messages of each signature to occur simultaneously with the last message of a previous signature, then the resulting protocol has two messages in effect, just like the 2018 2-of-n DKLs protocol.

Comparison to DKLs18. Compared to the 2-of-n DKLs protocol from 2018, our new protocol requires pipelining (and thus the storage of intermediate state) in order to achieve a two-round structure. We note that in the two party-case the downside implied by this is minimal: the stored state is exclusively pairwise, just like the stored state already required by the OT-extension protocol that is used to realize $\mathcal{F}_{\mathsf{RVOLE}}$. On the other hand, our new protocol realizes a *standard* threshold signing functionality, whereas the 2018 DKLs protocol realizes a

weaker functionality that allows the adversary to bias R, and that is only known to be equivalent to the standard functionality in the generic group model. Moreover, our protocol is statistically secure, whereas the 2018 protocol requires a reduction to the computational Diffie-Hellman assumption in the signing curve, and a reduction to the forgery game for ECDSA. Finally, our protocol improves upon the efficiency of the 2018 protocol. We do not make use of zero-knowledge proofs of knolwedge, whereas the 2018 DKLs protocol does; this allows us to avoid the overhead of straight-line extractable proofs [Fis05, Ks22], which is by far the most computationally-expensive component of the 2018 protocol. We also improve upon the bandwidth of the 2018 protocol: the chosen-input multiplication subprotocols used in that work require a total of $4\kappa + 4\lambda_{\mathsf{s}}$ correlated OT instances, half of which have a payload size of 2κ and half of which have a correlation size of 4κ. Realizing the randomized VOLE instances required by our new protocol via the VOLE protocol proposed in section 6 requires a total of $2\kappa + 4\lambda_{\mathsf{s}}$ correlated OT instances, all of which have a correlation size of 3κ. When $\kappa = 256$ and $\lambda_{\mathsf{s}} = 80$, as is common in practice, this yields a 38% savings in the bandwidth due to the OT payload *alone*. This improvement is independent of improvements due to new OT-extension techniques, and independent of an additional bandwidth-saving optimization that we introduce in our VOLE construction. As discussed in section 8, the overall bandwidth reduction achieved by our protocol when these improvements are considered is 57.3%.

4 Proof of Security for t-Party ECDSA

In section 1, we stated our security theorem:

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

There is, however, one caveat we must address when formalizing the above theorem. The UC model officially captures only a computational notion of security, and if it is extended to permit unbounded environments and adversaries, then a problem arises when considering protocols that realize reactive functionalities such as $\mathcal{F}_{\mathsf{ECDSA}}$: if each invocation implies a statistically negligible chance of distinguishing the real and ideal worlds, but the environment is allowed exponentially-many invokations, then, the overall probability of distinguishing such a protocol from its functionality becomes noticeable. To avoid this, we enforce explicit (but arbitrary) polynomial bounds on both the number of parties and the number of times the honest parties may be invoked, while allowing the environment to be otherwise unbounded. Thus we have the following formal theorem:

Theorem 4.1 (Formal Threshold ECDSA Security Theorem)**.** *For every malicious adversary* $\mathcal{A}$ *that statically corrupts up to* $t-1$ *parties, there exists a PPT*

simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}$ *that uses* $\mathcal{A}$ *as a black box, such that for every environment* $\mathcal{Z}$ *and every pair of polynomials* μ, ν*, if* $\mu(\lambda)$ *bounds the number of times* $\mathcal{Z}$ *invokes any honest party, then*

$$\left\{ \begin{array}{c} \textsc{Real}_{\pi_{\mathsf{ECDSA}}(\mathcal{G},n,t),\ \mathcal{A},\mathcal{Z}}(\lambda, z) : \\ \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda) \end{array} \right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\\ t\in[2,n],\ z\in\{0,1\}^*}}$$

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

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

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

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

Setup:

1. On receiving $(\texttt{keygen}, \mathsf{sid})$ from $\mathcal{P}_i$ for some $i \in \mathbf{P}^*$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, send
   - $(\texttt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_i$
   - $(\texttt{keygen-req}, \mathsf{sid}, i)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$
2. On receiving $(\texttt{init-req}, \mathsf{sid}, j)$ for some $j \in [n] \setminus \mathbf{P}^*$ directly from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$, send $(\texttt{keygen-req}, \mathsf{sid}, j)$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.
3. On receving $(\texttt{abort}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, send $(\texttt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.
4. On receiving $(\texttt{adv-poly}, \mathsf{sid}, \{\check{p}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\check{P}(j)\}_{j\in\mathbf{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, compute $\check{P}(i) := \check{p}(i) \cdot G$ for $i \in [n] \setminus \mathbf{P}^*$

and abort if $\check{P}(i)$ is not a polynomial of degree $t-1$ over $\mathbb{G}$.

5. On receiving

   - $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$
   - $(\texttt{adv-poly}, \mathsf{sid}, \{\check{p}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\check{P}(j)\}_{j\in\mathbf{P}^*})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$

   sample $\hat{p}_i(j) \leftarrow \mathbb{Z}_q$ uniformly for all $j \in \mathbf{P}^*$ and $i \in [n]\setminus\mathbf{P}^*$ and compute

   $$P(j) := \check{P}(j) + \sum_{i\in[n]\setminus\mathbf{P}^*} \hat{p}_i(j) \cdot G$$

   for every $j \in \mathbf{P}^*$. Let $P(0) := \mathsf{pk}$. If there are fewer than t points fixed on the polynomial P, then fix $t-1-|\mathbf{P}^*|$ points uniformly. Now, interpreting P as a polynomial of degree $t-1$ over $\mathbb{G}$, interpolate $P(i)$ for $i \in [n] \setminus \mathbf{P}^*$ and compute $\hat{P}(i) := P(i) - \check{p}(i) \cdot G$. Send $(\texttt{hon-poly}, \mathsf{sid}, \{\hat{P}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\hat{p}(j)\}_{j\in\mathbf{P}^*})$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, and store $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}, \{P(i)\}_{i\in[n]})$ in memory.

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

7. On receiving $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_j$ for $j \in \mathbf{P}^*$, store $(\texttt{sk-released}, \mathsf{sid}, j)$ in memory.

8. Initialize the blacklist for sid to be empty.

Signing:

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

   $$\begin{aligned} \mathbf{P}\|\mathsf{sigid}' &:= \mathsf{sigid} \quad \text{such that } |\mathbf{P}| = t \\ \mathbf{C} &:= \mathbf{P} \cap \mathbf{P}^* \\ \mathbf{H} &:= \mathbf{P} \setminus \mathbf{C} \\ \mathbf{P}^{-k} &:= \mathbf{P} \setminus \{k\} \qquad \text{for } k \in \mathbf{P} \end{aligned}$$

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

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

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

11. Upon receiving

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

    for some $i \in \mathbf{C}$ and some $j \in \mathbf{H}$, send $(\texttt{sample}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathsf{sigid}, \boldsymbol{\chi}_{i,j})$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$.

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

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

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

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

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

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

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

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

and send to $\mathcal{P}_i$

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

13. Upon satisfying satisfying steps 9 and 12 for every $j \in \mathbf{H}$ and receiving $(\texttt{abort}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid})$ directly from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$ for some $i \in \mathbf{C}$ and some $j \in \mathbf{H}$, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ to all corrupt parties on behalf of $\mathcal{P}_j$, *send the* `fail` *message on behalf of* $\mathcal{P}_j$ *at the corresponding point in all concurrent signing sessions involving* $\mathcal{P}_j$ *and* $\mathcal{P}_i$, append (j, i) to the blacklist for sid, and ignore all future instructions pertaining to the signature ID sigid.

14. Upon satisfying satisfying steps 9 and 12 for every $j \in \mathbf{H}$ and receiving

    - $(\texttt{multiply}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{a}^{\mathsf{u}}_{i,j}, \mathbf{a}^{\mathsf{v}}_{i,j}\})$ from $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$
    - $(\texttt{adv-share}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{c}^{\mathsf{u}}_{i,j}, \mathbf{c}^{\mathsf{v}}_{i,j}\})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$

    for some $i \in \mathbf{C}$ and some $j \in \mathbf{H}$, send $(\texttt{share}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathsf{sigid}, \{\mathbf{c}^{\mathsf{u}}_{i,j}, \mathbf{c}^{\mathsf{v}}_{i,j}\})$ to $\mathcal{P}_i$ on behalf of $\mathcal{F}_{\mathsf{RVOLE}}(q, 2)$.

15. Upon satisfying satisfying steps 9 and 12 for every $j \in \mathbf{H}$ and receiving

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

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

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

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

  then send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid}, k)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ for every $k \in \mathbf{H}$, send $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$ to all corrupt parties on behalf of $\mathcal{P}_j$, *send the* `fail` *message on behalf of* $\mathcal{P}_j$ *at the corresponding point in all concurrent signing sessions involving* $\mathcal{P}_j$ *and* $\mathcal{P}_i$, append (j, i) to the blacklist for sid, and ignore all future instructions pertaining to the signature ID sigid.

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

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

  then compute

$$\begin{aligned}
u_j &\coloneqq r_j \cdot \sum_{i \in \mathbf{C}} \boldsymbol{\psi}_{i,j} + \delta^{\mathsf{u}}_j \\
&\quad + \sum_{i \in \mathbf{C}} \begin{pmatrix} (\boldsymbol{\phi}_j - \boldsymbol{\psi}_{j,i}) \cdot \mathbf{a}^{\mathsf{u}}_{i,j} \\ + \boldsymbol{\chi}_{i,j} \cdot r_j - \mathbf{c}^{\mathsf{u}}_{i,j} - \mathbf{d}^{\mathsf{u}}_{i,j} \end{pmatrix} \\
v_j &\coloneqq \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \boldsymbol{\psi}_{i,j} + \delta^{\mathsf{v}}_j \\
&\quad + \sum_{i \in \mathbf{C}} \begin{pmatrix} (\boldsymbol{\phi}_j - \boldsymbol{\psi}_{j,i}) \cdot \mathbf{a}^{\mathsf{v}}_{i,j} \\ + \boldsymbol{\chi}_{i,j} \cdot \mathsf{sk}_j - \mathbf{c}^{\mathsf{v}}_{i,j} - \mathbf{d}^{\mathsf{v}}_{i,j} \end{pmatrix} \\
w_j &\coloneqq \mathsf{SHA2}(m_h) \cdot \phi_j + r^{\mathsf{x}} \cdot v_j
\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Our sequence of hybrid experiments begins with the real world

$$\mathcal{H}_0 = \left\{ \begin{array}{l} \textsc{Real}_{\pi_{\mathsf{ECDSA}}(\mathcal{G},n,t),\, (\lambda, z) :} \\ \qquad\qquad\qquad\qquad \mathcal{G} \leftarrow \mathsf{GrpGen}(1^\lambda) \end{array} \right\}_{\substack{\lambda \in \mathbb{N},\ n \in [2, \nu(\lambda)], \\ t \in [2,n],\ z \in \{0,1\}^*}}$$

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

Hybrid $\mathcal{H}_1$. This hybrid experiment replaces all of the individual honest parties and ideal functionalities in $\mathcal{H}_0$ with a single simulator machine $\mathcal{S}$ that runs

their code and interacts with the adversary, environment, and corrupt parties on their behalf. Since $\mathcal{S}$ interacts with the adversarial entities on behalf of the ideal functionalities, it learns any values they receive or that are defined by their internal state (for example, the value $\hat{p}(0)$ that defines the honest parties' contributions to the secret key sk). This is a purely syntactical change, and so it must be the case that $\mathcal{H}_1 = \mathcal{H}_0$.

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

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

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

but instead checks whether $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G = \mathbf{R}_{i,j}$ and $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G = \mathbf{pk}_{i,j}$ and $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$, as specified in step 15 of $\mathcal{S}_{\mathsf{ECDSA}}$.

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

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

we know that if the consistency checks evaluated in $\mathcal{H}_2$ pass, then the checks in $\mathcal{H}_1$ also pass. We also know that if $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G = \mathbf{R}_{i,j}$ and $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G = \mathbf{pk}_{i,j}$, then the checks in both hybrids pass if and only if $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j} = \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G$ and $\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} = \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G$. Thus, the adversary can only distinguish the two by setting $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G \neq \mathbf{R}_{i,j}$ or $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G \neq \mathbf{pk}_{i,j}$ for some corrupt $\mathcal{P}_j$ and contriving to pass the consistency check in $\mathcal{H}_1$, while failing the check (with certainty) in $\mathcal{H}_2$. Because $\mathbf{R}_{i,j}$ is fixed, there is exactly one value of $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}$ that will satisfy equation 1 for any assignment of $\boldsymbol{\chi}_{j,i}$ and $\mathbf{d}^{\mathsf{u}}_{j,i}$. Since $\boldsymbol{\chi}_{j,i}$ and $\mathbf{d}^{\mathsf{u}}_{j,i}$ are uniformly sampled and information-theoretically hidden from the adversary at the time that it must commit to $\boldsymbol{\Gamma}^{\mathsf{u}}_{i,j}$, the probability that the adversary sends this value is exactly $1/q$ if $\mathbf{a}^{\mathsf{u}}_{i,j} \cdot G \neq \mathbf{R}_{i,j}$. A similar argument implies that the adversary has a $1/q$ probability of satisfying equation 2 if $\mathbf{a}^{\mathsf{v}}_{i,j} \cdot G \neq \mathbf{pk}_{i,j}$.

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

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

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

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

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

Hybrid $\mathcal{H}_4$. This hybrid behaves identically to $\mathcal{H}_3$, except when the environment triggers a single signing instance with ID sigid between a group containing two or more honest parties, but uses inconsistent messages with the honest parties. Suppose $\mathcal{P}_h$ is the last honest party in the group to be activated by the environment. In $\mathcal{H}_4$, $\mathcal{S}$ replaces m_j with m_h in the calculations of every honest $\mathcal{P}_j$ for $j \in \mathbf{H} \setminus \{h\}$. If there exists some $j \in \mathbf{H} \setminus \{h\}$ such that environment sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{sigid}, m_j)$ to $\mathcal{P}_j$ and $\mathsf{SHA2}(m_j) \neq \mathsf{SHA2}(m_h)$, then $\mathcal{S}$ samples $w_h \leftarrow \mathbb{Z}_q$ instead of calculating w_h per the instructions in step 8 of π_{ECDSA} as in $\mathcal{H}_3$, always outputs $(\mathtt{failure}, \mathsf{sid}, \mathsf{sigid})$ to the environment on behalf of all honest parties, and ignores all future messages with the same sigid.

In $\mathcal{H}_3$, honest parties fail if they do not receive a valid signature as output, and we have argued in the context of $\mathcal{H}_3$ that a group of signers always receives a valid signature as output when their messages have the same image

under $\mathsf{SHA2}$ and nobody deviates from the protocol. In $\mathcal{H}_3$, $\mathcal{S}$ always calculates $w_h := \mathsf{SHA2}(m_h) \cdot \phi_h + r^{\mathsf{x}} \cdot v_h$ and $w_j := \mathsf{SHA2}(m_j) \cdot \phi_j + r^{\mathsf{x}} \cdot v_j$. We make three observations. First, the leftmost terms of these equations are the *only* constituent parts of the final signature that depend upon m_h or m_j. Second, $\mathcal{S}$ effectively samples w_h and w_j uniformly subject to a condition on their sum, because v_j and v_h depend linearly on $\mathbf{c}^{\mathsf{v}}_{j,h} + \mathbf{d}^{\mathsf{v}}_{j,h}$ and $\mathbf{c}^{\mathsf{v}}_{h,j} + \mathbf{d}^{\mathsf{v}}_{h,j}$ respectively, and the latter values are sampled uniformly subject to a condition on their sum. Third, due to similar linear dependencies upon $\mathbf{c}^{\mathsf{u}}_{j,h} + \mathbf{d}^{\mathsf{u}}_{j,h}$ and $\mathbf{c}^{\mathsf{u}}_{h,j} + \mathbf{d}^{\mathsf{u}}_{h,j}$, we can conclude that u_j and u_h commit $\mathcal{S}$ to the *sum* of ϕ_j and ϕ_h, but $\mathcal{S}$ still has a degree of freedom in choosing the individual values.[7]

Summing and rewriting, we have

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

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

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

Hybrid $\mathcal{H}_5$. The behavior of this hybrid differs from $\mathcal{H}_4$ when the environment triggers a signing instance among a group of parties, some of whom are corrupt. In $\mathcal{H}_5$, if the honest parties receive messages that have identical images under $\mathsf{SHA2}$, then $\mathcal{S}$ uses the $\mathsf{ECDSASign}$ to generate the signature, and embeds it into the protocol by altering the code of one of the honest signers. Specifically, $\mathcal{S}$

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

follows the code of the honest parties on their behalves until step 7 of π_{ECDSA}. Whichever honest party reaches this step *last* is designated $\mathcal{P}_h$ (as before), and if the honest parties have messages with identical images under $\mathsf{SHA2}$, then the code of $\mathcal{P}_h$ is replaced for the remainder of the protocol.

When the time comes for $\mathcal{S}$ to decommit $\mathcal{P}_h$'s contribution to the nonce on behalf of $\mathcal{F}_{\mathsf{Com}}$, rather that decommitting the value of R_h that was committed by $\mathcal{P}_h$ in step 6 of π_{ECDSA}, $\mathcal{S}$ instead computes

$$\mathsf{sk} := \sum_{i \in \mathbf{C}} \mathbf{a}^{\mathsf{v}}_{i,h} + \sum_{i \in \mathbf{H}} \mathsf{sk}_i$$

samples $(s, r^{\mathsf{x}}) \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$, reconstructs R from the x-coordinate r^{x}, computes

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

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

Next, if the consistency checks (specified in step 15 of $\mathcal{S}_{\mathsf{ECDSA}}$) pass, then $\mathcal{S}$ uses its knowledge of the corrupt parties' inputs and outputs from $\mathcal{F}_{\mathsf{RVOLE}}$ to predict the values of u_i and w_i that all of the parties apart from $\mathcal{P}_h$ would use, if no parties cheated. We will denote the *sum* of these predicted values as $\hat{u}$ and $\hat{w}$ respectively. These values depend upon ϕ_i for $i \in \mathbf{C}$, which might have been used inconsistently in interactions with the different honest parties, if the corrupt parties have misbehaved. $\mathcal{S}$ defines the true value of ϕ_i to be the value implied by the interaction between $\mathcal{P}_i$ and $\mathcal{P}_h$. Note that $\phi_i - \boldsymbol{\psi}_{i,h} - \boldsymbol{\chi}_{i,h} = 0$ by definition, which implies that there is no discrepancy between the values $\mathcal{P}_h$ computes if the corrupt parties follow the protocol and the values it computes if they misbehave, conditioned on the fact that the consistency check passes.

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

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

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

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

$$\hat{v} := \sum_{i \in \mathbf{C}} \left(\mathbf{a}^{\mathsf{v}}_{i,h} \cdot \Big(\sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + \boldsymbol{\psi}_{h,i} \Big) + \mathbf{c}^{\mathsf{v}}_{i,h} + \mathbf{d}^{\mathsf{v}}_{i,h} \right)$$
$$+ \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta^{\mathsf{v}}_j + \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right)$$
$$\hat{w} := \mathsf{SHA2}(m) \cdot \sum_{k \in \mathbf{P}^{\text{-}h}} \phi_k + r^{\mathsf{x}} \cdot \hat{v}$$

Note that because $\mathbf{c}^{\mathsf{u}}$, $\mathbf{d}^{\mathsf{u}}$, $\mathbf{c}^{\mathsf{v}}$, and $\mathbf{d}^{\mathsf{v}}$ are uniformly sampled subject to constraints upon their component-wise sums, δ^{u}_j and δ^{v}_j are uniform when considered indepently of the view of $\mathcal{P}_h$. Note also that when the consistency check passes for $\mathcal{P}_h$, we can be sure that $\mathbf{a}^{\mathsf{v}}_{i,h} \cdot G = \mathbf{pk}_{i,h}$ for every $i \in \mathbf{C}$, which implies that $\mathsf{sk} \cdot G = \mathsf{pk}$. By the same argument as we made in the context of $\mathcal{H}_3$, these constraints imply that the output of the protocol in $\mathcal{H}_4$ is a valid ECDSA signature on m under pk and R when all parties follow the protocol.

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

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

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

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

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

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

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

We now have

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

and by transitivity we also have

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

where the total statistical distance between $\mathcal{H}_6$ and $\mathcal{H}_0$ is upper-bounded by $\nu(\lambda) \cdot (\nu(\lambda) + \mu(\lambda))/q$, such that ν and μ are polynomials and q is exponential in λ. We can conclude that theorem 4.1 holds. □

5 Random Vector OLE from Random OT

In section 3.1, we introduce the random VOLE functionality $\mathcal{F}_{\mathsf{RVOLE}}$ and here we describe a protocol to realize it.

Recall that Vector OLE is a one-sided vectorization of Oblivious Linear Evaluation, i.e. the secure computation of additive shares of a product. This is in contrast to two-sided vectorization, which is sometimes referred to as *Batch*. The term *VOLE* was coined by Applebaum et al. [ADI+17]; protocols realizing OLE, VOLE, etc. have traded under many names, including simply *secure multiplication* or *multiplicative-to-additive conversion*. A large and diverse set of approaches for constructing such protocols is available, and most variants of the functionality can be realized via most approaches.

Of particular note are folkloric approaches based on additively homomorphic encryption. Such approaches are implicitly used in many other threshold ECDSA protocols that do not make explicit use of an OLE functionality. For example, Lindell [Lin17], Lindell and Nof [LN18], Gennarro

and Goldfeder [GG18], and Canetti et al. [CGG+20] all implicitly construct OLE protocols[8] using Paillier's encryption scheme [Pai99]. Castagnos et al. [CCL+20, CCL+23] make similar use of homomorphic encryption from class groups [CL15]. It is also known how to construct VOLE from code-theoretic assumptions [ADI+17, CRR21] and lattice assumptions [dJV21, BDM22], and in the amortized setting the celebrated *silent* random VOLE protocol family [BCGI18, BCG+19, SGRR19, YWL+20] has bandwidth costs almost independent of the vector length.

It should be stressed that it is possible to realize $\mathcal{F}_{\mathsf{RVOLE}}$ using *any* of these techniques, and therefore the security of our threshold ECDSA protocol can be founded on *any* of the above assumptions while achieving a wide variety of potential performance tradeoffs. However, we do make a specific recommendation that we believe leads to concretely efficient results in many use-cases.

Our VOLE protocol ultimately derives from one of the oldest OLE techniques, Gilboa's OT-based multiplication protocol [Gil99], which uses a simple schoolbook technique to destructure a multiplication in an arbitrary field of size q into $|q|$-many one-bit multiplications, which are performed using oblivious transfer. Gilboa's protocol is secure only against semi-honest adversaries. Keller et al. [KOS16] implicitly constructed a version with malicious security as a component in a protocol for generating Beaver triples. Two years later, Doerner et al. [DKLs18] explicitly constructed a malicious-secure variant of Gilboa's OLE protocol. To our knowledge, this was the first two-message maliciously secure OLE protocol. In a follow-up work, Doerner et al. [DKLs19] presented batched version of their protocol with somewhat reduced bandwidth requirements, at the cost of an additional round. Later, Haitner et al. [HMRT22] made another moderate reduction in the bandwidth cost of this technique, again requiring three rounds, but only realizing a weakened functionality that does not ensure correct outputs are produced in the event that the protocol completes without an abort.

In this work, we revisit and improve Doerner et al.'s malicious OLE technique. We construct a (random) VOLE with two rounds, which has concretely lower bandwidth requirements than either their original or their batched construction, and also lower bandwidth requirements than the protocol of Haitner et al. under some parameterizations, even though it realizes a stronger functionality in fewer rounds. As with prior iterations of this idea, our protocol is based upon vectors of OT instances, which we model using an OT-extension functionality (although this can be realized using a vector of normal OT functionalities, if desired). We choose to base our scheme upon an *endemic* OT-extension functionality [MR19]; that is, upon vectors of OT instances wherein the adversary is always allowed to determine the outputs of any corrupt party. This is the weakest well-defined OT functionality, and all other varieties of OT trivially imply it. The specific functionality is as follows:

[8] Though not necessarily ones that achieve security independently of the context in which they are embedded.

Functionality 5.1. $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{X}, \ell_{\mathsf{OT}})$**: Endemic OT Extension**

This functionality interacts with two active participants, $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$, who we refer to as Alice and Bob, and with the ideal adversary $\mathcal{S}$.

OT Extension: On receiving $(\texttt{choose}, \mathsf{sid}, \boldsymbol{\beta})$ from Bob, if sid is fresh, and $\boldsymbol{\beta} \in \{0,1\}^{\ell_{\mathsf{OT}}}$, then:

- If Alice is corrupt, then send $(\texttt{bob-chosen}, \mathsf{sid})$ to $\mathcal{S}$ and wait for $(\texttt{alice-messages}, \mathsf{sid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1)$ in response such that $\boldsymbol{\alpha}^0 \in \mathbb{X}^{\ell_{\mathsf{OT}}}$ and $\boldsymbol{\alpha}^1 \in \mathbb{X}^{\ell_{\mathsf{OT}}}$
- If Alice is corrupt, then wait for $(\texttt{bob-message}, \mathsf{sid}, \boldsymbol{\gamma})$ from $\mathcal{S}$ such that $\boldsymbol{\gamma} \in \mathbb{X}^{\ell_{\mathsf{OT}}}$, and then for $i \in [\ell_{\mathsf{OT}}]$ let $\boldsymbol{\alpha}_i^{\boldsymbol{\beta}_i} := \boldsymbol{\gamma}$ and sample $\boldsymbol{\alpha}_i^{1-\boldsymbol{\beta}_i} \leftarrow \mathbb{X}$.
- If either party is corrupt and $\mathcal{S}$ sends $(\texttt{abort}, \mathsf{sid})$, then forward this message to both parties and perform no further instructions related to the session with ID sid.
- If neither party is corrupt, then sample $\boldsymbol{\alpha}^0 \leftarrow \mathbb{X}^{\ell_{\mathsf{OT}}}$ and $\boldsymbol{\alpha}^1 \leftarrow \mathbb{X}^{\ell_{\mathsf{OT}}}$.

Finally send $(\texttt{choice-made}, \mathsf{sid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1)$ to Alice and $(\texttt{chosen}, \mathsf{sid}, \boldsymbol{\gamma})$ to Bob.

We suggest to realize the foregoing functionality via the recent SoftSpoken random OT-extension protocol of Roy [Roy22], because it requires only one round in the Random Oracle model (under a simple modification that we will describe in section 5.1), performs well in the non-amortized setting given a fast one-time setup procedure, and assumes only ideal OT. Most other OT and OT-extension protocols are also suitable, and in particular we expect *silent* OT-extension [BCG+19] to be the most efficient option when a large number of signatures must be generated at once. Next, we give our protocol. Since it derives strongly from Doerner et al. [DKLs18, DKLs19], the proof of security that we present in section 6 is inspired by theirs.

Protocol 5.2. $\pi_{\mathsf{RVOLE}}(q, \ell)$**: OT-Based Random Vector OLE**

This protocol is parameterized by the vector length ℓ and modulus q of the group $\mathbb{Z}_q$ over which multiplication is to be performed. Let $\kappa = |q|$ and for convenience let $\xi = \kappa + 2\lambda_{\mathsf{s}}$ and $\rho = \lceil \kappa / \lambda_{\mathsf{c}} \rceil$. This protocol makes use of a public *gadget vector* $\mathbf{g} \leftarrow \mathbb{Z}_q^{\xi}$,[a] and it invokes the $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$ functionality and the non-programmable global random oracle $\mathsf{RO}_{\mathbb{X}}$, which has a paramatric range specified by its subscript.

Sampling:

1. When Bob receives $(\texttt{sample}, \mathsf{sid})$ from the environment, where sid is a fresh session ID, Bob samples a set of uniform OT choice bits $\boldsymbol{\beta} \leftarrow \{0,1\}^{\xi}$ and calculates his random "input" $b := \langle \mathbf{g}, \boldsymbol{\beta} \rangle$ and then he sends

$(\texttt{choose}, \mathsf{sid}, \boldsymbol{\beta})$ to $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$, and outputs $(\texttt{sample}, \mathsf{sid}, b)$ to the environment.

2. Upon receiving $\left(\texttt{choice-made}, \mathsf{sid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1\right)$ from $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$, Alice outputs $(\texttt{ready}, \mathsf{sid})$ to the environment.

Multiplication:

3. When Alice has received both $(\texttt{multiply}, \mathsf{sid}, \mathbf{a})$ from the environment and $\left(\texttt{choice-made}, \mathsf{sid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1\right)$ from $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$,[b] she computes her output share, samples a set of check values, derandomizes her OT messages using her inputs as correlations, calculates a challenge via the Fiat-Shamir heuristic, and compresses her response via the random oracle

$$\mathbf{c} \coloneqq \left\{ -\sum_{j\in[\xi]} \mathbf{g}_j \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{i\in[\ell]}$$
$$\hat{\mathbf{a}} \leftarrow \mathbb{Z}_q^{\rho}$$
$$\tilde{\mathbf{a}} \coloneqq \left\{ \left\{ \boldsymbol{\alpha}^0_{j,i} - \boldsymbol{\alpha}^1_{j,i} + \mathbf{a}_i \right\}_{i\in[\ell]} \| \left\{ \boldsymbol{\alpha}^0_{j,\ell+k} - \boldsymbol{\alpha}^1_{j,\ell+k} + \hat{\mathbf{a}}_i \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$
$$\boldsymbol{\theta} \coloneqq \mathsf{RO}_{\mathbb{Z}_q^{\ell\times\rho}}(\mathsf{sid}, \tilde{\mathbf{a}})$$
$$\boldsymbol{\eta} \coloneqq \left\{ \hat{\mathbf{a}}_k + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \mathbf{a}_i \right\}_{k\in[\rho]}$$
$$\boldsymbol{\mu} \coloneqq \left\{ \left\{ \boldsymbol{\alpha}^0_{j,\ell+k} + \sum_{i\in[\ell]} \boldsymbol{\theta}_{k,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$
$$\mu \coloneqq \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu})$$

and after this, she sends $(\texttt{multiply}, \mathsf{sid}, \tilde{\mathbf{a}}, \boldsymbol{\eta}, \mu)$ to Bob and outputs outputs $(\texttt{share}, \mathsf{sid}, \mathbf{c})$ to the environment.

4. When Bob receives $(\texttt{multiply}, \mathsf{sid}, \tilde{\mathbf{a}}, \boldsymbol{\eta}, \mu)$ from Alice and $(\texttt{chosen}, \mathsf{sid}, \boldsymbol{\gamma})$ from $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$, Bob computes

$$\boldsymbol{\theta} \coloneqq \mathsf{RO}_{\mathbb{Z}_q^{\ell\times\rho}}(\mathsf{sid}, \tilde{\mathbf{a}})$$
$$\dot{\mathbf{d}} \coloneqq \left\{ \left\{ \boldsymbol{\gamma}_{j,i} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,i} \right\}_{i\in[\ell]} \right\}_{j\in[\xi]}$$
$$\hat{\mathbf{d}} \coloneqq \left\{ \left\{ \boldsymbol{\gamma}_{j,\ell+k} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,\ell+k} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

$$\boldsymbol{\mu}' := \left\{ \left\{ \hat{\mathbf{d}}_{j,k} + \sum_{i \in [\ell]} \boldsymbol{\theta}_{i,k} \cdot \dot{\mathbf{d}}_{j,i} - \boldsymbol{\beta}_j \cdot \boldsymbol{\eta}_k \right\}_{k \in [\rho]} \right\}_{j \in [\xi]}$$

and checks whether $\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu}')$. If this check fails[c] then Bob aborts. If it passes, then Bob computes

$$\mathbf{d} := \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot \dot{\mathbf{d}}_{j,i} \right\}_{i \in [\ell]}$$

and outputs $(\texttt{share}, \mathsf{sid}, \mathbf{d})$ to the environment.

[a] This vector can be sampled by Bob and reused, or it can be a CRS reused by *multiple* Bobs in different instances of the protocol.

[b] If $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$ aborts instead of delivering a `choice-made` message, then Alice aborts to the environment.

[c] Or if $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$ aborts instead of delivering a `chosen` message.

Derandomizing our Random VOLE. Finally, we note that it is trivial to construct standard chosen-input VOLE using the random VOLE functionality that the foregoing protocol realizes: Bob simply transmits to Alice the difference between his true input and b, and for each of Alice's inputs, she adds the product of this difference and her input to her corresponding output. This requires no additional rounds, and allows our performance improvements to be applied to other protocols that use the DKLs OLE protocols, including Chen et al.'s [CCD+20] RSA modulus sampling protocol and Doerner et al.'s threshold BBS+ protocol [DKL+23], in which context it saves more than half of the bandwidth cost.

5.1 One-Message SoftSpokenOT in the ROM

As previously mentioned, we must modify the SoftSpokenOT protocol in order to achieve our round count target. The modification we make is well-explored and does not alter the security analysis of the protocol in a significant way. It requires the use of a non-programmable random oracle. An analogous technique is used by the KOS OT-extension protocol [KOS15], and by our own VOLE construction and its progenitors, and it was previously suggested to apply this modification to SoftSpokenOT by Doerner et al. [DKL+23]. In this subsection, we describe SoftSpokenOT in terms of the notation of Keller et al. [KOS15], who describe a version of SoftSpokenOT in a recent update to their work.

As written, SoftSpokenOT requires three rounds to create a batch of extensions, after the initial one-time setup is complete. These rounds comprise a statistical check with the form of a sigma protocol: first the OT-extension receiver commits, then the sender transmits a challenge, then the receiver responds. However, the protocol does not require rewinding to achieve secu-

rity, and the simulator does not use this test to extract anything from the receiver's view. As a result, we can apply the Fiat-Shamir transform using a non-programmable (and even non-observable) global random oracle, and the protocol retains UC security. That is, in the notation of Keller et al., we compute $(\chi_1, \ldots, \chi_m) \xleftarrow{\$} \mathsf{H}(\mathbf{u}^1, \ldots, \mathbf{u}^\kappa)$. This reduces the protocol to a single round from receiver to sender (after the initial setup). Note that because the adversary can attempt to find a convenient challenge by brute force under this optimization, each occurrence of the statistical parameter (s in the notation of Keller et al.) in the original protocol must be replaced by the computational security parameter (κ in the notation of Keller et al.).

6 Proof of Security for OT-Based VOLE

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

Theorem 1.2 (Informal Random VOLE Security Theorem)**.** *In the $\mathcal{F}_{\mathsf{EOTE}}$-hybrid non-programmable global random oracle model, $\pi_{\mathsf{RVOLE}}(q, \ell)$ UC-realizes $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$ against a PPT malicious adversary that statically corrupts no more than one party.*

In this section, we give our security theorem formally, and provide a proof.

Theorem 6.1 (Formal OT-Based Random VOLE Security Theorem)**.** *For every malicious PPT adversary $\mathcal{A}$ that statically corrupts either $\mathcal{P}_\mathsf{A}$ or $\mathcal{P}_\mathsf{B}$, there exists a PPT simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE}}$ that uses $\mathcal{A}$ as a black box, such that for every PPT environment $\mathcal{Z}$ and every polynomial ν,*

$$\left\{\textsc{Real}_{\pi_{\mathsf{RVOLE}}(q,\ell),\mathcal{A},\mathcal{Z}}\left(\lambda, z\right)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]\,:\,q \text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$
$$\approx_{\mathrm{c}} \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{RVOLE}}(q,\ell),\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE}}(q,\ell),\mathcal{Z}}\left(\lambda, z\right)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]\,:\,q \text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$

Proof. Lemma 6.2 in section 6.1 asserts that there exists a simulator $\mathcal{S}_{\mathsf{RVOLE\text{-}Alice}}$ under which theorem 6.1 holds, conditioned on $\mathcal{A}$ corrupting only $\mathcal{P}_\mathsf{A}$. Lemma 6.11 in section 6.2 asserts the existence of $\mathcal{S}_{\mathsf{RVOLE\text{-}Bob}}$ such that theorem 6.1 holds when $\mathcal{A}$ corrupts only $\mathcal{P}_\mathsf{B}$. The conjunction of these statements yields theorem 6.1. □

6.1 Simulating Against Alice

Lemma 6.2 (OT-Based VOLE Security against Alice)**.** *For every malicious PPT adversary $\mathcal{A}$ that statically corrupts* only *$\mathcal{P}_\mathsf{A}$, there exists a simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}$ that uses $\mathcal{A}$ as a black box, such that for every PPT environment $\mathcal{Z}$*

and every polynomial ν*,*

$$\left\{\text{REAL}_{\pi_{\mathsf{RVOLE}}(q,\ell),\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]:q\text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$
$$\approx_{\text{c}} \left\{\text{IDEAL}_{\mathcal{F}_{\mathsf{RVOLE}}(q,\ell),\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell),\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]:q\text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$

Proof. We begin by presenting $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}$, after which we prove the lemma via a sequence if hybrid experiments.

Simulator 6.3. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell)$**: Random VOLE against Alice**

This simulator is parameterized by the vector length ℓ and modulus q of the group $\mathbb{Z}_q$ over which linear evaluation is to be performed. The simulator has oracle access to the adversary $\mathcal{A}$ that statically corrupts $\mathcal{P}_{\mathsf{A}}$ (i.e. the party playing Alice) *only*, and emulates for it an instance of the protocol $\pi_{\mathsf{RVOLE}}(q,\ell)$ involving the parties $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell)$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ on behalf of $\mathcal{P}_{\mathsf{A}}$, and in the exeriment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and $\mathcal{P}_{\mathsf{A}}$ on behalf of $\mathcal{P}_{\mathsf{B}}$ and on behalf of the ideal oracle $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$. As in $\pi_{\mathsf{RVOLE}}(q,\ell)$, let $\kappa=|q|$ and $\xi=\kappa+2\lambda_{\mathsf{s}}$ and $\rho=\lceil\kappa/\lambda_{\mathsf{c}}\rceil$, let $\mathbf{g}\leftarrow\mathbb{Z}_q^{\xi}$ be a uniform gadget vector, and let $\mathsf{RO}_{\mathbb{X}}$ be a non-programmable global random oracle with a paramatric range specified by its subscript.

Sampling:

1. On receiving $(\texttt{ready},\mathsf{sid})$ from $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ on behalf of $\mathcal{P}_{\mathsf{A}}$, send $(\texttt{bob-chosen},\mathsf{sid})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$ and wait to receive $(\texttt{alice-messages},\mathsf{sid},\boldsymbol{\alpha}^0,\boldsymbol{\alpha}^1)$ in response, then send $\left(\texttt{choice-made},\mathsf{sid},\boldsymbol{\alpha}^0,\boldsymbol{\alpha}^1\right)$ to $\mathcal{P}_{\mathsf{A}}$ on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$. If $\mathcal{A}$ sends `abort` instead of `alice-messages`, then send $(\texttt{abort},\mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ and perform no further instructions related to the session with ID sid.

Multiplication:

2. On receiving $(\texttt{multiply},\mathsf{sid},\tilde{\mathbf{a}},\boldsymbol{\eta},\mu)$ from Alice, check the table of queries that have been made to the global random oracle for a query on a value $\boldsymbol{\mu}\in\mathbb{Z}_q^{\xi\times\rho}$ such that $\mu=\mathsf{RO}_{\{0,1\}^{2\lambda_{\mathsf{c}}}}(\mathsf{sid},\boldsymbol{\mu})$. If there is not exactly one such value, then send $(\texttt{abort},\mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ and perform no further instructions related to the session with ID sid. Otherwise, continue to the next step.

3. Using the value $\boldsymbol{\mu}$ extracted in the previous step, compute two vectors of offsets, relative to the expectation:

$$\boldsymbol{\theta}\coloneqq\mathsf{RO}_{\mathbb{Z}_q^{\ell\times\rho}}(\mathsf{sid},\tilde{\mathbf{a}})$$

$$\boldsymbol{\delta}^{\mu} := \left\{ \left\{ \boldsymbol{\mu}_{j,k} - \boldsymbol{\alpha}^0_{j,\ell+k} - \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

$$\boldsymbol{\delta}^{\eta} := \left\{ \left\{ \begin{array}{l} \boldsymbol{\alpha}^1_{j,\ell+k} - \boldsymbol{\alpha}^0_{j,\ell+k} + \tilde{\mathbf{a}}_{j,\ell+k} - \boldsymbol{\eta}_k \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{j,i} \cdot \left(\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} + \tilde{\mathbf{a}}_{j,i}\right) \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

Note that if Alice has behaved honestly, then $\boldsymbol{\delta}^{\mu}$ and $\boldsymbol{\delta}^{\eta}$ will contain only zeros. Sample $\boldsymbol{\beta}^* \leftarrow \{0,1\}^{\xi}$ and if

$$\bigvee_{\substack{k\in[\rho] \\ j\in[\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} \neq \boldsymbol{\beta}^*_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}$$

then send $(\mathtt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ and perform no further instructions related to the session with ID sid. Otherwise, continue to the next step.

4. Find $j^* \in [\xi]$ such that $\boldsymbol{\delta}^{\eta}_{j,k} = 0$ for every $k \in [\rho]$, and let this index define Alice's true input:

$$\mathbf{a} := \left\{ \boldsymbol{\alpha}^1_{j^*,i} - \boldsymbol{\alpha}^0_{j^*,i} + \tilde{\mathbf{a}}_{j^*,i} \right\}_{i\in[\ell]}$$

If there are multiple candidate values for j^*, then choose the smallest one. If there exists no j^* satisfying these conditions then send $(\mathtt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ and perform no further instructions related to the session with ID sid. Otherwise, continue to the next step.

5. Compute Alice's output, and the offset induced into Bob's output by any "undetected" cheats, and subtract the latter from the former

$$\mathbf{c} := \left\{ -\sum_{j\in[\xi]} \mathbf{g}_j \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{i\in[\ell]}$$

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

$$\mathbf{c}^* := \left\{ \mathbf{c}_i - \boldsymbol{\delta}^d_i \right\}_{i\in[\ell]}$$

and then send

- $(\texttt{multiply}, \mathsf{sid}, \mathbf{a})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$ on behalf of Alice.
- $(\texttt{alice-share}, \mathsf{sid}, \mathbf{c}^*)$ directly to $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$.

Our sequence of hybrid experiments begins with the real world

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

and proceeds by gradually replacing the code of the real $\mathcal{P}_{\mathsf{B}}$ with elements of the simulator until $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}$ is fully implemented and the experiment is the ideal one.

Hybrid $\mathcal{H}_1$. This hybrid experiment replaces the honest party $\mathcal{P}_{\mathsf{B}}$ and the ideal functionality $\mathcal{F}_{\mathsf{EOTE}}$ in $\mathcal{H}_0$ with a single simulator machine $\mathcal{S}$ that runs their code and interacts with the adversary, environment, and corrupt party $\mathcal{P}_{\mathsf{A}}$ on their behalf. Since $\mathcal{S}$ interacts with the adversarial entities on behalf of $\mathcal{F}_{\mathsf{EOTE}}$ it learns any values that $\mathcal{F}_{\mathsf{EOTE}}$ receives or that are defined by its internal state. Furthermore, $\mathcal{S}$ is permitted to observe queries to the random oracle, but not to program the output. These changes are purely syntactical, and so it must be the case that $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In this hybrid, $\mathcal{S}$ aborts on behalf of Bob upon receiving $(\texttt{multiply}, \mathsf{sid}, \tilde{\mathbf{a}}, \boldsymbol{\eta}, \mu)$ from Alice if there does not exist exactly one value $\boldsymbol{\mu} \in \mathbb{Z}_q^{\xi\times\rho}$ such that $\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu})$. This can be determined efficiently by scanning the table of random oracle queries.

We claim that $\mathcal{H}_2 \approx_c \mathcal{H}_1$. Consider the two distinguishing cases: first, that no query of the form $\mathbb{Z}_q^{\xi\times\rho}$ has ever yielded the output μ, and second, that more than one has. In the first case, we know $\mathcal{S}$ will query a value that has never before been queried, per Bob's code in step 4 of the protocol, and so the probabilty that $\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu}')$ is exactly $2^{-2\lambda_c}$. The second case implies that the environment has found a collision in the random oracle. The probability that a collision occurs within the first Q queried values is the sum of the probabilities that each new attempt produces a collision; specifically, the probability of a collision is within the first Q queried values is

$$0 + 2^{-2\lambda_c} + 2\cdot 2^{-2\lambda_c} + 3\cdot 2^{-2\lambda_c} + \ldots + (Q-1)\cdot 2^{-2\lambda_c} \leq Q^2\cdot 2^{-2\lambda_c - 1}$$

and by union bound, we have a total distinguishing probability no greater than $Q^2\cdot 2^{-2\lambda_c-1} + 2^{-2\lambda_c}$. Since the environment can make only polynomially many random oracle queries, this probability is negligible in λ_c.

Hybrid $\mathcal{H}_3$. In this hybrid, $\mathcal{S}$ does not condition an abort on whether $\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu}')$ as specified in step 4 of the protocol. Instead, $\mathcal{S}$ computes implements step 3 of $\mathcal{S}_{\mathsf{RVOLE\text{-}Alice}}$, but uses Bob's actual choice bits $\boldsymbol{\beta}$ instead of

$\boldsymbol{\beta}^*$. More precisely, in $\mathcal{H}_3$, $\mathcal{S}$ computes

$$\boldsymbol{\theta} := \mathsf{RO}_{\mathbb{Z}_q^{\ell\times\rho}}(\mathsf{sid}, \tilde{\mathbf{a}})$$

$$\boldsymbol{\delta}^{\mu} := \left\{ \left\{ \boldsymbol{\mu}_{j,k} - \boldsymbol{\alpha}^0_{j,\ell+k} - \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

$$\boldsymbol{\delta}^{\eta} := \left\{ \left\{ \begin{array}{l} \boldsymbol{\alpha}^1_{j,\ell+k} - \boldsymbol{\alpha}^0_{j,\ell+k} + \tilde{\mathbf{a}}_{j,\ell+k} - \boldsymbol{\eta}_k \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{j,i} \cdot \left(\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} + \tilde{\mathbf{a}}_{j,i}\right) \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

and then aborts if

$$\bigvee_{\substack{k\in[\rho]\\ j\in[\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} \neq \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}$$

We claim that $\mathcal{H}_3 \approx_{\mathrm{s}} \mathcal{H}_2$. Observe that in $\mathcal{H}_2$, the exeriment aborts with certainty if there is not exactly one correctly formatted preimage $\boldsymbol{\mu}$ of μ under the random oracle. Because $\boldsymbol{\mu}$ is the only (correctly formatted) preimage of μ, we have

$$\Pr\left[\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_{\mathrm{c}}}}(\mathsf{sid}, \boldsymbol{\mu}') : \boldsymbol{\mu} \neq \boldsymbol{\mu}'\right] = 2^{-2\lambda_{\mathrm{c}}}$$

or, in other words, conditioned on $\boldsymbol{\mu}$ being well-defined, $\mathcal{H}_2$ aborts with overwhelming probability if $\boldsymbol{\mu} \neq \boldsymbol{\mu}'$, and if $\boldsymbol{\mu} = \boldsymbol{\mu}'$ then it does not abort. We will show that $\mathcal{H}_3$ aborts if and only if $\boldsymbol{\mu} \neq \boldsymbol{\mu}'$, again conditioned on $\boldsymbol{\mu}$ being well-defined.

We can subtract

$$\left\{ \left\{ \boldsymbol{\alpha}^0_{j,\ell+k} + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

from both sides of $\boldsymbol{\mu} \neq \boldsymbol{\mu}'$ to yield

$$\left\{ \left\{ \boldsymbol{\mu}_{j,k} - \boldsymbol{\alpha}^0_{j,\ell+k} - \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

$$\neq \left\{ \left\{ \boldsymbol{\mu}'_{j,k} - \boldsymbol{\alpha}^0_{j,\ell+k} - \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

and then substituting in the definitions of $\boldsymbol{\mu}'$ and $\boldsymbol{\delta}^{\mu}$ we have

$$\boldsymbol{\delta}^{\mu} \neq \left\{ \left\{ \hat{\mathbf{d}}_{j,k} + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \dot{\mathbf{d}}_{j,i} - \boldsymbol{\alpha}^0_{j,\ell+k} - \boldsymbol{\beta}_j \cdot \boldsymbol{\eta}_k - \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

after which, substituting the definitions of $\dot{\mathbf{d}}$ and $\hat{\mathbf{d}}$ and then the definition of $\boldsymbol{\gamma}$ and simplifying yields

$$\boldsymbol{\delta}^{\mu} \neq \left\{ \left\{ \boldsymbol{\beta}_j \cdot \left(\begin{array}{l} \boldsymbol{\alpha}^1_{j,\ell+k} - \boldsymbol{\alpha}^0_{j,\ell+k} + \tilde{\mathbf{a}}_{j,\ell+k} - \boldsymbol{\eta}_k \\ + \sum_{i \in [\ell]} \boldsymbol{\theta}_{i,k} \cdot (\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} + \tilde{\mathbf{a}}_{j,i}) \end{array} \right) \right\}_{k \in [\rho]} \right\}_{j \in [\xi]}$$

which is equivalent to

$$\bigvee_{\substack{k \in [\rho] \\ j \in [\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} \neq \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}$$

and it follows from this that $\mathcal{H}_3 \approx_{\mathrm{s}} \mathcal{H}_2$ with statistical distance $2^{-2\lambda_{\mathrm{c}}}$.

Hybrid $\mathcal{H}_4$. In this hybrid, $\mathcal{S}$ implements step 4 of $\mathcal{S}_{\mathsf{RVOLE\text{-}Alice}}$. That is, it finds the smallest $j^* \in [\xi]$ such that $\boldsymbol{\delta}^{\eta}_{j,k} = 0$ for every $k \in [\rho]$, and defines

$$\mathbf{a} \coloneqq \left\{ \boldsymbol{\alpha}^1_{j^*,i} - \boldsymbol{\alpha}^0_{j^*,i} + \tilde{\mathbf{a}}_{j^*,i} \right\}_{i \in [\ell]}$$

if such a j^* exists, or aborts if no such j^* exists.

We claim that $\mathcal{H}_4 \approx_{\mathrm{s}} \mathcal{H}_3$. Consider an adversary that distinguishes the two hybrids; specifically, consider an adversary that fixes $\boldsymbol{\delta}^{\eta}$ such that for every $j \in [\xi]$ there exists some $k \in [\rho]$ such that $\boldsymbol{\delta}^{\eta}_{j,k} \neq 0$. If this adversary avoids an abort in $\mathcal{H}_3$, then

$$\bigwedge_{\substack{k \in [\rho] \\ j \in [\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} = \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}$$

or, in other words, that it has exactly guessed the entire vector $\boldsymbol{\beta}$ of Bob's choice bits. Notice that Bob's output b is the *only* value that depends upon $\boldsymbol{\beta}$ in the view of the environment at the point that $\boldsymbol{\delta}^{\eta}$ and $\boldsymbol{\delta}^{\eta}$ are fixed.

We now construct an alternative experiment. In our alternative experiment behaves exactly like $\mathcal{H}_4$, except that $\mathcal{S}$ samples $b \leftarrow \mathbb{Z}_q$ uniformly rather than deriving it from $\boldsymbol{\beta}$, and our alternative experiment terminates when Bob receives $(\mathtt{multiply}, \mathsf{sid}, \tilde{\mathbf{a}}, \boldsymbol{\eta}, \mu)$ from Alice, fixing $\boldsymbol{\delta}^{\eta}$ and $\boldsymbol{\delta}^{\eta}$. We make a sequence of claims:

Claim 6.4. *In our alternative experiment,*

$$\Pr\left[\bigwedge_{\substack{k \in [\rho] \\ j \in [\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} = \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k} \,\middle|\, \bigwedge_{j \in [\xi]} \boldsymbol{\delta}^{\eta}_{j,*} \neq \{0\}^{\rho} \right] \leq 2^{-\xi}$$

Our first claim holds because $\boldsymbol{\beta}$ is perfectly information-theoretically hidden from the adversary in our alternative experiment: no value derived from it enters the adversary's view. When $\boldsymbol{\delta}^{\eta}$ contains no zero values, the adversary's best strategy for avoiding the abort condition is to guess.

Claim 6.5. *At the moment of our alternative experiment's termination, the statistical difference between the adversary's view in our alternative experiment and its view at the equivalent point in $\mathcal{H}_4$ is at most $2^{-\lambda_s}$.*

Our second claim holds because the only distinction between the two is the way in which b is calculated. In our alternative experiment it is uniform and independent of all other values, whereas in $\mathcal{H}_4$ we have $b = \langle \mathbf{g}, \boldsymbol{\beta} \rangle$ and no other values depend upon $\boldsymbol{\beta}$. Because (by construction) $\kappa < \xi$, we can apply part 2 of proposition 1.1 of Impagliazzo and Naor [IN96] to conclude that the statistical difference between the two experiments is at most $2^{(\kappa-\xi)/2} = 2^{-\lambda_s}$.

Claim 6.6. *In $\mathcal{H}_4$,*

$$\Pr\left[\bigwedge_{\substack{k\in[\rho]\\ j\in[\xi]}} \boldsymbol{\delta}^{\mu}_{j,k} = \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k} \,\middle|\, \bigwedge_{j\in[\xi]} \boldsymbol{\delta}^{\eta}_{j,*} \neq \{0\}^{\rho}\right] \leq 2^{-\xi} + 2^{-\lambda_s}$$

Our final claim holds by conjunction of claims 6.5 and 6.4, and it implies that $\mathcal{H}_4 \approx_s \mathcal{H}_3$ with statistical distance at most $2^{-\xi} + 2^{-\lambda_s}$.

Hybrid $\mathcal{H}_5$. This hybrid changes the way in which $\mathcal{S}$ computes an output for Bob. In $\mathcal{H}_4$, Bob's output $\mathbf{d}$ was calculated via his protocol code. In $\mathcal{H}_5$, if no abort occurs, then $\mathcal{S}$ uses the index j^* that was initially defined in $\mathcal{H}_4$ to extract an ideal input for Alice

$$\mathbf{a} \coloneqq \left\{\boldsymbol{\alpha}^1_{j^*,i} - \boldsymbol{\alpha}^0_{j^*,i} + \tilde{\mathbf{a}}_{j^*,i}\right\}_{i\in[\ell]}$$

and in addition, $\mathcal{S}$ calculates the output $\mathbf{c}$ that Alice will emit if she behaves honestly, and uses this to compute the ideal output $\mathbf{d}^*$ that Bob would emit if Alice behaved honestly. $\mathcal{S}$ then computes the vector $\boldsymbol{\delta}^d$ of additive offsets induced into Bob's outputs by Alice's cheats, and applies them to Bob's ideal output in order to calculate his actual output $\mathbf{d}$

$$\begin{aligned}
\mathbf{c} &\coloneqq \left\{-\sum_{j\in[\xi]} \mathbf{g}_j \cdot \boldsymbol{\alpha}^0_{j,i}\right\}_{i\in[\ell]} \\
\mathbf{d}^* &\coloneqq \{b \cdot \mathbf{a}_i - \mathbf{c}_i\}_{i\in[\ell]} \\
\boldsymbol{\delta}^d &\coloneqq \left\{\sum_{j\in[\xi]} \mathbf{g}_j \cdot \boldsymbol{\beta}_j \cdot (\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} + \tilde{\mathbf{a}}_{j,i} - \mathbf{a}_i)\right\}_{i\in[\ell]} \\
\mathbf{d} &\coloneqq \left\{\mathbf{d}^*_i + \boldsymbol{\delta}^d_i\right\}_{i\in[\ell]}
\end{aligned}$$

This effectively implements step 5 of $\mathcal{S}_{\mathsf{RVOLE\text{-}Alice}}$, but using Bob's actual choice bits $\boldsymbol{\beta}$ instead of $\boldsymbol{\beta}^*$.

We claim that $\mathcal{H}_5$ and $\mathcal{H}_4$ are identically distributed. Observe that by substituting the definitions of $\boldsymbol{\delta}^d$ and $\mathbf{d}^*$ into the equation that defines $\mathbf{d}$ in $\mathcal{H}_5$, we have

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

and since $b = \langle \mathbf{g}, \boldsymbol{\beta} \rangle$ we can plug in the definition of $\mathbf{c}$ and simplify to arrive at

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

wherafter we can substitute $\boldsymbol{\gamma}_{j,i} + (\boldsymbol{\beta}_j - 1) \cdot \boldsymbol{\alpha}^0_{j,i} = \boldsymbol{\beta}_j \cdot \boldsymbol{\alpha}^1_{j,i}$, which follows from the code of $\mathcal{F}_{\mathsf{EOTE}}$, to yield

$$\mathbf{d} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot \left(\boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,i} + \boldsymbol{\gamma}_{j,i} \right) \right\}_{i \in [\ell]} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot \dot{\mathbf{d}}_{j,i} \right\}_{i \in [\ell]}$$

which is precisely the equation used to calculate Bob's output in $\mathcal{H}_4$.

Hybrid $\mathcal{H}_6$. This hybrid is identical to $\mathcal{H}_5$, except that in $\mathcal{H}_6$, $b \leftarrow \mathbb{Z}_q$ is sampled uniformly, rather than being computed as $b := \langle \mathbf{g}, \boldsymbol{\beta} \rangle$. Note that while b is now independent of $\boldsymbol{\beta}$, $\mathcal{S}$ still aborts based upon $\boldsymbol{\beta}$, and Bob's output $\mathbf{d}$ depends upon both.

We claim that $\mathcal{H}_6 \approx_{\mathrm{c}} \mathcal{H}_5$. Recall that in both, Alice's true input is defined to be

$$\mathbf{a} := \left\{ \boldsymbol{\alpha}^1_{j^*,i} - \boldsymbol{\alpha}^0_{j^*,i} + \tilde{\mathbf{a}}_{j^*,i} \right\}_{i \in [\ell]}$$

where $j^* \in [\xi]$ is the smallest number such that $\boldsymbol{\delta}^{\eta}_{j,k} = 0$ for every $k \in [\rho]$. We now define some additional values:

$$\begin{aligned}
\hat{\mathbf{a}} &:= \left\{ \boldsymbol{\alpha}^1_{j^*,\ell+k} - \boldsymbol{\alpha}^0_{j^*,\ell+k} + \tilde{\mathbf{a}}_{j^*,\ell+k} \right\}_{k \in [\rho]} \\
\boldsymbol{\delta}^a &:= \left\{ \left\{ \boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} + \tilde{\mathbf{a}}_{j,i} - \mathbf{a}_i \right\}_{i \in [\ell]} \right\}_{j \in [\xi]} \\
\boldsymbol{\delta}^{\hat{a}} &:= \left\{ \left\{ \boldsymbol{\alpha}^1_{j,\ell+k} - \boldsymbol{\alpha}^0_{j,\ell+k} + \tilde{\mathbf{a}}_{j,\ell+k} - \hat{\mathbf{a}}_k \right\}_{k \in [\rho]} \right\}_{j \in [\xi]} \\
\boldsymbol{\delta}^{\dot{d}} &:= \left\{ \left\{ \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^a \right\}_{i \in [\ell]} \right\}_{j \in [\xi]} \\
\boldsymbol{\Delta} &:= \{ j \in [\xi] : \boldsymbol{\delta}^{\eta}_{j,*} \neq \{0\}^{\rho} \}
\end{aligned}$$

For clarity, $\hat{\mathbf{a}}$ is defined to be Alice's true check vector, $\boldsymbol{\delta}^a$ and $\boldsymbol{\delta}^{\hat{a}}$ are vectors of the offsets by which Alice has deviated from her true input and check vector

respectively, and $\boldsymbol{\delta}^{\dot{d}}$ is a vector of the offsets that her deviations induce into Bob's output. Notice that

$$\boldsymbol{\delta}^{d} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot \boldsymbol{\delta}^{\dot{d}}_{j,i} \right\}_{i \in [\ell]}$$

Finally, $\boldsymbol{\Delta}$ indexes all of the elements in $\boldsymbol{\beta}$ that *might* influence the probability that the experiment aborts; an abort may also happen with certainty independent of these elements. We now make a sequence of claims, from which we will construct our argument that $\mathcal{H}_6 \approx_{\mathrm{c}} \mathcal{H}_5$:

Claim 6.7. *If for some $j \in [\xi]$ there exists $i \in [\ell]$ such that $\boldsymbol{\delta}^{a}_{j,i} \neq 0$, then $\Pr[j \in \boldsymbol{\Delta}] \geq 1 - Q \cdot 2^{-\lambda_{\mathsf{c}}}$, where Q is the number of random oracle queries made by the adversary.*

Suppose $j \notin \boldsymbol{\Delta}$. This implies that $\boldsymbol{\delta}^{\eta}_{j,*} = \{0\}^{\rho}$, and for every $k \in [\rho]$ we have

$$\begin{pmatrix} \boldsymbol{\alpha}^{1}_{j,\ell+k} - \boldsymbol{\alpha}^{0}_{j,\ell+k} + \tilde{\mathbf{a}}_{j,\ell+k} \\ + \sum_{i \in [\ell]} \boldsymbol{\theta}_{j,i} \cdot \left(\boldsymbol{\alpha}^{1}_{j,i} - \boldsymbol{\alpha}^{0}_{j,i} + \tilde{\mathbf{a}}_{j,i} \right) \end{pmatrix}$$
$$= \begin{pmatrix} \boldsymbol{\alpha}^{1}_{j^*,\ell+k} - \boldsymbol{\alpha}^{0}_{j^*,\ell+k} + \tilde{\mathbf{a}}_{j^*,\ell+k} \\ + \sum_{i \in [\ell]} \boldsymbol{\theta}_{j^*,i} \cdot \left(\boldsymbol{\alpha}^{1}_{j^*,i} - \boldsymbol{\alpha}^{0}_{j^*,i} + \tilde{\mathbf{a}}_{j^*,i} \right) \end{pmatrix}$$

Recall that $\boldsymbol{\alpha}^0$ and $\boldsymbol{\alpha}^1$ are fixed and then the adversary computes $\boldsymbol{\theta} = \mathsf{RO}_{\mathbb{Z}_q^{\ell \times \rho}}(\mathsf{sid}, \tilde{\mathbf{a}})$. For any given adversarial choice of $\tilde{\mathbf{a}}$, the probability over the coins of the random oracle that the above equality holds simultaneously for every $k \in [\rho]$ is $1/q^{\rho} \leq 2^{-\lambda_{\mathsf{c}}}$. If we let Q be the total number of random oracle queries made by the adversary over the course of the experinment, then the claim follows.

Claim 6.8. *In $\mathcal{H}_5$, if $\mathbf{a}$ is well-defined, then the probability that the experiment does not abort is at most $2^{-|\boldsymbol{\Delta}|}$.*

For every $j \in \boldsymbol{\Delta}$, the experiment aborts if $\boldsymbol{\delta}^{\mu}_{j,*} \neq \{\boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}\}_{k \in [\rho]}$, which happens with probability at least $1/2$.[9] Thus the probability of an abort is at least $1 - 2^{-|\boldsymbol{\Delta}|}$ and taking the complement yields claim 6.8.

Claim 6.9. *The probability that $\mathcal{H}_6$ aborts and the probability that $\mathcal{H}_5$ aborts have an absolute difference no greater than $2^{-\lambda_{\mathsf{s}}}$.*

The above claim follows from the same argument as claim 6.5.

Claim 6.10. *If $|\boldsymbol{\Delta}| \leq 2\lambda_{\mathsf{s}}$ and neither hybrid aborts, then the statistical difference between $\mathcal{H}_6$ and $\mathcal{H}_5$ is upper-bounded by $2^{-(\xi - \kappa - |\boldsymbol{\Delta}|)/2} + \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}$, where Q is the number of random oracle queries made by the adversary.*

[9] The experiment aborts with probability 1 if $\boldsymbol{\delta}^{\mu}_{j,*} \neq \boldsymbol{\delta}^{\eta}_{j,*}$ and $\boldsymbol{\delta}^{\mu}_{j,*} \neq \{0\}^{\rho}$.

To aid our analysis of this claim, we will first develop an alternate view of $\mathcal{H}_6$ and $\mathcal{H}_5$. Suppose that in $\mathcal{H}_5$, we write

$$b^{\Delta} := \sum_{j \in \boldsymbol{\Delta}} \mathbf{g}_j \cdot \boldsymbol{\beta}_j \qquad \text{and} \qquad b^{\bar{\Delta}} := \sum_{j \in [\xi] \setminus \boldsymbol{\Delta}} \mathbf{g}_j \cdot \boldsymbol{\beta}_j \qquad \text{and} \qquad b := b^{\Delta} + b^{\bar{\Delta}}$$

which is, of course, only a syntactical change. Now in $\mathcal{H}_6$ we write

$$b^{\Delta} := \sum_{j \in \boldsymbol{\Delta}} \mathbf{g}_j \cdot \boldsymbol{\beta}_j \qquad \text{and} \qquad b^{\bar{\Delta}} \leftarrow \mathbb{Z}_q \qquad \text{and} \qquad b := b^{\Delta} + b^{\bar{\Delta}}$$

and again, we have not changed the distribution of $\mathcal{H}_6$; $b$remains uniform. Assuming $|\boldsymbol{\Delta}| < \xi - \kappa = 2\lambda_{\mathsf{s}}$, we can again apply part 2 of proposition 1.1 of Impagliazzo and Naor [IN96] to conclude that the statistical difference between $b^{\bar{\Delta}}$ in $\mathcal{H}_6$ and $\mathcal{H}_5$ is at most $2^{-(\xi-\kappa-|\boldsymbol{\Delta}|)/2}$.

It remains to show that *no variables* in either experiment depend upon $\boldsymbol{\beta}_j$ for $j \in [\xi] \setminus \boldsymbol{\Delta}$, except indirectly via b. For every $j \in [\xi] \setminus \boldsymbol{\Delta}$ we have by definition that $\boldsymbol{\delta}^{\eta}_{j,*} = \{0\}^{\rho}$, which implies that $\boldsymbol{\delta}^{\mu}_{j,k} \neq \boldsymbol{\beta}_j \cdot \boldsymbol{\delta}^{\eta}_{j,k}$ is satisfied (or not) for all $k \in [\rho]$ independently of the value of $\boldsymbol{\beta}_j$. Similarly, claim 6.7 implies that for each $j \in [\xi] \setminus \boldsymbol{\Delta}$, we have $\Pr[\boldsymbol{\delta}^{a}_{j,*} \neq \{0\}^{\ell}] \leq Q \cdot 2^{-\lambda_{\mathsf{c}}}$. Taking the compliment of the union bound over all $j \in [\xi] \setminus \boldsymbol{\Delta}$, we have

$$\Pr\left[\bigwedge_{j \in [\xi] \setminus \boldsymbol{\Delta}} \boldsymbol{\delta}^{a}_{j,*} = \{0\}^{\ell}\right] \geq 1 - \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}$$

This implies that $\boldsymbol{\delta}^{\dot{d}}$ and consequently $\boldsymbol{\delta}^{d}$ and are independent of $\boldsymbol{\beta}_j$ for $j \in [\xi] \setminus \boldsymbol{\Delta}$ with probability no less than $1 - \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}$. If they are independent, then the statistical distance between the two hybrids is at most $2^{-(\xi-\kappa-|\boldsymbol{\Delta}|)/2}$, and if they are dependent, then we assume the two hybrids can be distinguished with probability 1. Claim 6.10 follows.

Finally, we are ready to complete our argument that $\mathcal{H}_6 \approx_{\mathsf{c}} \mathcal{H}_5$. Per claim 6.8, the probability that $\mathcal{H}_5$ completes without an abort is at most $2^{-|\boldsymbol{\Delta}|}$. Per claim 6.9, the distribution of aborts in $\mathcal{H}_6$ is statistically indistinguishable from the distribution of aborts in $\mathcal{H}_5$. The we can pair the input tapes for $\mathcal{H}_6$ and $\mathcal{H}_5$ in such a way that the fraction of pairs that cause both hybrids to compete without aborting with is at most $2^{-|\boldsymbol{\Delta}|} - 2^{-\lambda_{\mathsf{s}}}$, and the fraction of pairs that cause both to abort is at least $1 - 2^{-|\boldsymbol{\Delta}|} - 2^{-\lambda_{\mathsf{s}}}$.

The only variable that distinguishes an aborted instance of $\mathcal{H}_6$ from an aborted instance of $\mathcal{H}_5$ is b. A simple application of part 2 of proposition 1.1 of Impagliazzo and Naor [IN96] (as already performed twice before in this proof) implies that the statistical distance between the distributions of the two hybrids is upper bounded by $2^{-(\xi-\kappa)/2} = 2^{-\lambda_{\mathsf{s}}}$, conditioned on both hybrids aborting.

The probability that $|\boldsymbol{\Delta}| > 2\lambda_{\mathsf{s}}$ and neither hybrid aborts is less than $2^{-2\lambda_{\mathsf{s}}} + 2^{-\lambda_{\mathsf{s}}}$. We assume that if $|\boldsymbol{\Delta}| > 2\lambda_{\mathsf{s}}$ and the experiment does not abort, then the adversary can distinguish.

Per claim 6.10, if $|\boldsymbol{\Delta}| \leq 2\lambda_{\mathsf{s}}$ and neither hybrid aborts, then the statistical difference between $\mathcal{H}_6$ and $\mathcal{H}_5$ is upper-bounded by $2^{-(\xi-\kappa-|\boldsymbol{\Delta}|)/2} + \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}$.

Putting the pieces together, the total statistical distance between the two hybrids is no greater than

$$\begin{aligned}
& \left(2^{-|\boldsymbol{\Delta}|} - 2^{-\lambda_{\mathsf{s}}}\right) \cdot \left(2^{-(\xi-\kappa-|\boldsymbol{\Delta}|)/2} + \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}\right) \\
& + \left(1 - 2^{-|\boldsymbol{\Delta}|} - 2^{-\lambda_{\mathsf{s}}}\right) \cdot 2^{-\lambda_{\mathsf{s}}} + 2 \cdot 2^{-\lambda_{\mathsf{s}}} + 2^{-2\lambda_{\mathsf{s}}} + 2^{-\lambda_{\mathsf{s}}} \\
< \;& 2^{-|\boldsymbol{\Delta}|} \cdot \left(2^{-(2\lambda_{\mathsf{s}}-|\boldsymbol{\Delta}|)/2} + \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}}\right) + 5 \cdot 2^{-\lambda_{\mathsf{s}}} \\
= \;& 2^{-\lambda_{\mathsf{s}}-|\boldsymbol{\Delta}|/2} + 2^{-|\boldsymbol{\Delta}|} \cdot \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}} + 5 \cdot 2^{-\lambda_{\mathsf{s}}} \\
< \;& \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}} + 6 \cdot 2^{-\lambda_{\mathsf{s}}}
\end{aligned}$$

which is negligible in λ_{c} so long as Q is at most polynomial in λ_{c}.

Hybrid $\mathcal{H}_7$. This final hybrid differs from $\mathcal{H}_6$ in the following way: $\mathcal{S}$ no longer acts on behalf of any honest parties. Instead, $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell)$ is fully implemented in $\mathcal{H}_7$, and the experiment now incorporates $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$. The honest parties run dummy-party code as is standard for ideal-world experiements in the UC model, and $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell)$ speaks to $\mathcal{F}_{\mathsf{RVOLE}}$ on behalf of corrupt parties.

The differences between $\mathcal{H}_7$ and $\mathcal{H}_6$ are essentially syntactical, which is to say that $\mathcal{H}_6 = \mathcal{H}_7$. The simulator's choice bit vector is now denoted $\boldsymbol{\beta}^*$ (whereas in $\mathcal{H}_6$ it was denoted $\boldsymbol{\beta}$). The simulator uses this vector to determine whether an abort will occur and to calculate the vector of offsets $\boldsymbol{\delta}^d$ induced into the output by Alice's inconsistencies, as before, but instead of adding these offsets to Bob's output, the simulator instead computes $\mathbf{c}^*_i := \mathbf{c}_i - \boldsymbol{\delta}^d_i$ for $i \in [\ell]$ and supplies $\mathbf{c}^*$ to $\mathcal{F}_{\mathsf{RVOLE}}$ in place of Alice's output $\mathbf{d}$. The functionality then computes $\mathbf{c}$ such that for every $i \in [\ell]$ it holds that $\mathbf{a}_i \cdot b = \mathbf{c}^*_i + \mathbf{d}_i$, and thus the relationship between $\mathbf{a}$, b, $\mathbf{c}$, and $\mathbf{d}$ is maintained.

We now have

$$\mathcal{H}_7 = \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{RVOLE}}(q,\ell),\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Alice}}(q,\ell),\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]\,:\,q \text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$

and by transitivity we also have

$$\mathcal{H}_7 \approx_{\mathrm{c}} \mathcal{H}_0 = \left\{\textsc{Real}_{\pi_{\mathsf{RVOLE}}(q,\ell),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]\,:\,q \text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$

where the total statistical distance between the $\mathcal{H}_7$ and $\mathcal{H}_0$ is no more than

$$\begin{aligned}
& \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}} + 6 \cdot 2^{-\lambda_{\mathsf{s}}} + 2^{-\xi} + 2^{-\lambda_{\mathsf{s}}} + 2^{-2\lambda_{\mathsf{c}}} + Q^2 \cdot 2^{-2\lambda_{\mathsf{c}}-1} + 2^{-2\lambda_{\mathsf{c}}} \\
< \;& \xi \cdot Q \cdot 2^{-\lambda_{\mathsf{c}}} + Q^2 \cdot 2^{-2\lambda_{\mathsf{c}}-1} + 8 \cdot 2^{-\lambda_{\mathsf{s}}} + 2 \cdot 2^{-2\lambda_{\mathsf{c}}}
\end{aligned}$$

and since the environment is PPT, we know that the total number of random oracle queries Q is at most polynomial in λ_{c}. We can conclude that Lemma 6.2 holds. □

6.2 Simulating Against Bob

Lemma 6.11 (OT-Based VOLE Security against Bob)**.** *For every malicious PPT adversary $\mathcal{A}$ that statically corrupts* only *$\mathcal{P}_{\mathsf{B}}$, there exists a simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Bob}}$ that uses $\mathcal{A}$ as a black box, such that for every PPT environment $\mathcal{Z}$ and every polynomial ν,*

$$\left\{\textsc{Real}_{\pi_{\mathsf{RVOLE}}(q,\ell),\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]:q\text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$
$$=\left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{RVOLE}}(q,\ell),\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Bob}}(q,\ell),\mathcal{Z}}(\lambda,z)\right\}_{\substack{\lambda\in\mathbb{N},\ n\in[2,\nu(\lambda)],\ t\in[2,n],\\ q\in[2^{\nu(\lambda)}]:q\text{ is prime},\ \ell\in[\nu(\lambda)],\ z\in\{0,1\}^*}}$$

Proof. We begin by presenting $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Bob}}$, after which we present an argument for the indistinguishability of the two experiments.

Simulator 6.12. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Bob}}(q,\ell)$**: Random VOLE against Bob**

This simulator is parameterized by the vector length ℓ and modulus q of the group $\mathbb{Z}_q$ over which linear evaluation is to be performed. The simulator has oracle access to the adversary $\mathcal{A}$ that statically corrupts $\mathcal{P}_{\mathsf{B}}$ (i.e. the party playing Bob) *only*, and emulates for it an instance of the protocol $\pi_{\mathsf{RVOLE}}(q,\ell)$ involving the parties $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RVOLE\text{-}Bob}}(q,\ell)$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ on behalf of $\mathcal{P}_{\mathsf{B}}$, and in the exeriment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and $\mathcal{P}_{\mathsf{B}}$ on behalf of $\mathcal{P}_{\mathsf{A}}$ and on behalf of the ideal oracle $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$. As in $\pi_{\mathsf{RVOLE}}(q,\ell)$, let $\kappa=|q|$ and $\xi=\kappa+2\lambda_{\mathsf{s}}$ and $\rho=\lceil\kappa/\lambda_{\mathsf{c}}\rceil$, let $\mathbf{g}\in\mathbb{Z}_q^{\xi}$ be an arbitrary non-zero gadget vector,[a] and let $\mathsf{RO}_{\mathbb{X}}$ be a non-programmable global random oracle with a paramatric range specified by its subscript.

Sampling:

1. On receiving $(\texttt{choose},\mathsf{sid},\boldsymbol{\beta})$ from Bob on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$, compute $b:=\langle\mathbf{g},\boldsymbol{\beta}\rangle$ and send
   - $(\texttt{sample},\mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ on behalf of Bob
   - $(\texttt{bob-sample},\mathsf{sid},b)$ directly to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$.
2. On receiving $(\texttt{bob-message},\mathsf{sid},\boldsymbol{\gamma})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$, send $(\texttt{chosen},\mathsf{sid},\boldsymbol{\gamma})$ to Bob on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho},\xi)$. If $\mathcal{A}$ sends `abort` instead of `bob-message`, then send $(\texttt{abort},\mathsf{sid})$ to $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$ and perform no further instructions related to the session with ID sid.

Multiplication:

3. After receiving both
   - $(\texttt{alice-multiplied},\mathsf{sid})$ from $\mathcal{F}_{\mathsf{RVOLE}}(q,\ell)$

- $(\texttt{bob-message}, \mathsf{sid}, \boldsymbol{\gamma})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{EOTE}}(\mathbb{Z}_q^{\ell+\rho}, \xi)$

sample $\tilde{\mathbf{a}} \leftarrow \mathbb{Z}_q^{\xi+\rho}$ and $\boldsymbol{\eta} \leftarrow \mathbb{Z}_q^{\rho}$, compute

$$\dot{\mathbf{d}} := \left\{ \{ \boldsymbol{\gamma}_{j,i} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,i} \}_{i \in [\ell]} \right\}_{j \in [\xi]}$$

$$\hat{\mathbf{d}} := \left\{ \{ \boldsymbol{\gamma}_{j,\ell+i} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,\ell+i} \}_{i \in [\rho]} \right\}_{j \in [\xi]}$$

$$\boldsymbol{\mu} := \left\{ \left\{ \hat{\mathbf{d}}_{j,k} + \sum_{i \in [\ell]} \boldsymbol{\theta}_{i,k} \cdot \dot{\mathbf{d}}_{j,i} - \boldsymbol{\beta}_j \cdot \boldsymbol{\eta}_k \right\}_{k \in [\rho]} \right\}_{j \in [\xi]}$$

$$\mu := \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu})$$

$$\mathbf{d} := \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot \dot{\mathbf{d}}_{j,i} \right\}_{i \in [\ell]}$$

and send

- $(\texttt{multiply}, \mathsf{sid}, \tilde{\mathbf{a}}, \boldsymbol{\eta}, \mu)$ to Bob on behalf of Alice.
- $(\texttt{bob-share}, \mathsf{sid}, \mathbf{d})$ directly to $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$.

[a] When simulating against a corrupt Bob, it is not necessary that $\mathbf{g}$ have any particular distribution.

The view of the environment in the two experiments is characterized by the view of the corrupt Bob, and by Alice's input $\mathbf{a}$, and her output $\mathbf{c}$. Bob's view is characterized by the vector $\boldsymbol{\beta}$, which determines b, by the vector $\boldsymbol{\gamma}$ that he receives from $\mathcal{F}_{\mathsf{EOTE}}$, and by the values $\tilde{\mathbf{a}}$, $\boldsymbol{\eta}$, and μ that he receives from Alice. We claim that the joint distribution of these values is identical in the real and ideal experiments.

In both experiments, $\mathbf{a}$ is chosen by the environment, $\boldsymbol{\beta}$ is chosen bythe corrupt Bob, and $\boldsymbol{\gamma}$ is supplied directly by the adversary. In both experiments, $\mu = \mathsf{RO}_{\{0,1\}^{2\lambda_c}}(\mathsf{sid}, \boldsymbol{\mu})$ for some $\boldsymbol{\mu}$, and $\boldsymbol{\theta} = \mathsf{RO}_{\mathbb{Z}_q^{\ell \times \rho}}(\mathsf{sid}, \tilde{\mathbf{a}})$. The values $\tilde{\mathbf{a}}$, $\boldsymbol{\eta}$, and $\boldsymbol{\mu}$, are calculated by the simulator in the ideal experiment and $\mathbf{c}$ is calculated by $\mathcal{F}_{\mathsf{RVOLE}}$; in the real experiment, these four values are all calculatd by honest Alice.

In the ideal experiment $\boldsymbol{\eta}$ is sampled uniformly per step 3 of $\mathcal{S}_{\mathsf{RVOLE\text{-}Bob}}$, whereas in the real experiment Alice calculates

$$\boldsymbol{\eta} := \left\{ \hat{\mathbf{a}}_k + \sum_{i \in [\ell]} \boldsymbol{\theta}_{i,k} \cdot \mathbf{a}_i \right\}_{k \in [\rho]} \tag{5}$$

per step 3 of π_{RVOLE}. $\hat{\mathbf{a}}$ is sampled uniformly by Alice, and none of the variables that we have analyzed so far depend upon it; therefore, from the adversary's perspective, $\boldsymbol{\eta}$ is uniform in both worlds when considered jointly with $(\mathbf{a}, \boldsymbol{\beta}, \boldsymbol{\gamma})$.

In the ideal experiment, $\tilde{\mathbf{a}}$ is sampled uniformly, whereas in the real experiment Alice calculates

$$\tilde{\mathbf{a}} := \left\{ \left\{ \boldsymbol{\alpha}^0_{j,i} - \boldsymbol{\alpha}^1_{j,i} + \mathbf{a}_i \right\}_{i\in[\ell]} \| \left\{ \boldsymbol{\alpha}^0_{j,\ell+k} - \boldsymbol{\alpha}^1_{j,\ell+k} + \hat{\mathbf{a}}_k \right\}_{k\in[\rho]} \right\}_{j\in[\xi]} \tag{6}$$

For every $j \in [\xi]$ and $k \in [\rho]$, $\boldsymbol{\alpha}^{1-\boldsymbol{\beta}_j}_{j,k}$ is uniformly sampled by $\mathcal{F}_{\mathsf{EOTE}}$ and information-theoretically hidden from the adversary in the real experiment, which implies that from the adversary's point of view, $\tilde{\mathbf{a}}$ is uniform in both experiments when considered jointly with $(\mathbf{a}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\eta})$. This also implies that the distribution of $\boldsymbol{\theta}$ is identical in both experiments.

In the ideal experiment, $\mathcal{S}_{\mathsf{RVOLE\text{-}Bob}}$ calculates

$$\boldsymbol{\mu} := \left\{ \left\{ \hat{\mathbf{d}}_{j,k} + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \dot{\mathbf{d}}_{j,i} - \boldsymbol{\beta}_j \cdot \boldsymbol{\eta}_k \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

and by substituting in the equations that $\mathcal{S}$ uses to calculate $\hat{\mathbf{d}}$ and $\dot{\mathbf{d}}$, we have

$$\boldsymbol{\mu} = \left\{ \left\{ \begin{array}{l} \boldsymbol{\gamma}_{j,\ell+k} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,\ell+k} \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot (\boldsymbol{\gamma}_{j,k} + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,k}) - \boldsymbol{\beta}_j \cdot \boldsymbol{\eta}_k \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]} \tag{7}$$

in the ideal world. In the real experiment, Alice calculates

$$\boldsymbol{\mu} := \left\{ \left\{ \boldsymbol{\alpha}^0_{j,\ell+k} + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

which expands to

$$\boldsymbol{\mu} = \left\{ \left\{ \begin{array}{l} (1-\boldsymbol{\beta}_j) \cdot \boldsymbol{\alpha}^0_{j,\ell+k} + \boldsymbol{\beta}_j \cdot \boldsymbol{\alpha}^0_{j,\ell+k} \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \left((1-\boldsymbol{\beta}_j) \cdot \boldsymbol{\alpha}^0_{j,i} + \boldsymbol{\beta}_j \cdot \boldsymbol{\alpha}^0_{j,i} \right) \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]}$$

and since $\boldsymbol{\gamma}_{j,i} = (1-\boldsymbol{\beta}_j) \cdot \boldsymbol{\alpha}^0_{j,i} + \boldsymbol{\beta}_j \cdot \boldsymbol{\alpha}^1_{j,i}$, if follows that

$$\boldsymbol{\mu} = \left\{ \left\{ \begin{array}{l} \boldsymbol{\gamma}_{j,\ell+k} + \boldsymbol{\beta}_j \cdot (\boldsymbol{\alpha}^0_{j,\ell+k} - \boldsymbol{\alpha}^1_{j,\ell+k}) \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \left(\boldsymbol{\gamma}_{j,i} + \boldsymbol{\beta}_j \cdot (\boldsymbol{\alpha}^0_{j,i} - \boldsymbol{\alpha}^1_{j,i}) \right) \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]} \tag{8}$$

in the real experiment. Now rearanging equation 6 to solve for $\boldsymbol{\alpha}^0_{j,i} - \boldsymbol{\alpha}^1_{j,i}$ and then substituting it into equation 8, we find that

$$\boldsymbol{\mu} = \left\{ \left\{ \begin{array}{l} \boldsymbol{\gamma}_{j,\ell+k} + \boldsymbol{\beta}_j \cdot (\tilde{\mathbf{a}}_{j,\ell+k} - \hat{\mathbf{a}}_k) \\ + \sum_{i\in[\ell]} \boldsymbol{\theta}_{i,k} \cdot \left(\boldsymbol{\gamma}_{j,i} + \boldsymbol{\beta}_j \cdot (\tilde{\mathbf{a}}_{j,i} - \mathbf{a}_i) \right) \end{array} \right\}_{k\in[\rho]} \right\}_{j\in[\xi]} \tag{9}$$

in the real experiment, and finally, substituting equation 5 into equation 9 yields equation 7, which is the ideal-world distribution. Thus the distribution of $\boldsymbol{\mu}$ is identical in the real and ideal experiments, when considered jointly with $(\mathbf{a}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \tilde{\mathbf{a}}, \boldsymbol{\theta})$.

The final variable is $\mathbf{c}$. In the ideal world, $\mathcal{F}_{\mathsf{RVOLE}}$ calculates $\mathbf{c}$, and plugging in the definitions of the values supplied to $\mathcal{F}_{\mathsf{RVOLE}}$ by $\mathcal{S}_{\mathsf{RVOLE\text{-}Bob}}$, we have

$$\mathbf{c} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot (\boldsymbol{\beta}_j \cdot \mathbf{a}_i + \boldsymbol{\beta}_j \cdot \tilde{\mathbf{a}}_{j,i} - \boldsymbol{\gamma}_{j,i}) \right\}_{i \in [\ell]} \tag{10}$$

in the ideal world. In the real world, Alice calculates

$$\mathbf{c} \coloneqq \left\{ -\sum_{j \in [\xi]} \mathbf{g}_j \cdot \boldsymbol{\alpha}^0_{j,i} \right\}_{i \in [\ell]} \tag{11}$$

and we can rewrite equation 6 and substitute the result into equation 11 to find that

$$\mathbf{c} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot (\mathbf{a}_i - \tilde{\mathbf{a}}_{j,i} - \boldsymbol{\alpha}^1_{j,i}) \right\}_{i \in [\ell]}$$

and adding $0 = (1 - \boldsymbol{\beta}_j) \cdot \boldsymbol{\alpha}^0_{j,i} + \boldsymbol{\beta}_j \cdot \boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\gamma}_{j,i}$ to this we find

$$\mathbf{c} = \left\{ \sum_{j \in [\xi]} \mathbf{g}_j \cdot (\mathbf{a}_i - \tilde{\mathbf{a}}_{j,i} + (\boldsymbol{\beta}_j - 1) \cdot (\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i}) - \boldsymbol{\gamma}_{j,i}) \right\}_{i \in [\ell]} \tag{12}$$

in the real experiment. Finally, rewriting equation 6 and plugging it into equation 12 yields equation 10. It follows that the distribution of $\mathbf{c}$ is identical in the real and ideal experiments when considered jointly with $(\mathbf{a}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \tilde{\mathbf{a}}, \boldsymbol{\theta}, \boldsymbol{\mu})$, and from this it follows that the two experiments are identically distributed overall, and lemma 6.11 holds. □

7 Relaxed Threshold Key Generation

In this section we discuss our key generation functionality, $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$, which was introduced in section 3.1. In section 7.1 we introduce a protocol to realize our functionality, and in section 7.2 we prove it to be perfectly secure.

We refer to our key generation functionality as *relaxed* because it differs in a crucial way from those used in prior works. Consider as an example of the typical approach the key generation functionality that accompanies the threshold BBS+ protocol Doerner et al. [DKL+23]. This functionality works in the "obvious" way: it samples a uniform secret key internally and outputs a corresponding public key to all parties, accepts Shamir shares of the secret key from the corrupt

parties, and outputs consistent shares to each of the honest parties. To prove that any protocol realizes this functionality, it is necessary for the simulator to program into the views of the corrupt parties a specific public key, and to extract their Shamir shares so that the functionality can produce consistent ones. In order to enable this, the three-round protocol that Doerner et al. specify includes a proof of knowledge for the contributions of each party. Such proofs of knowledge are a hallmark of simulation-secure key generation protocols in the dishonest-majority setting,[10] and in the context of universal composability, they must be straight-line extractable, which implies significant overhead in terms of both computation and communication due to the use of the Pass [Pas03], Fischlin [Fis05], or Kondi-shelat [Ks22] transforms.

Consider the above "obvious" key generation formulation. The outputs of the functionality to the honest parties are non-negotiable, since they will be forwarded directly to the environment in the UC model. The interface with the corrupt parties, on the other hand, is driven by the needs of the simulators: both the simulator for the key generation protocol, and also the simulator for any eventual signing protocol that uses the key generation functionality and requires knowledge of the adversary's shares in order to simulate signing.

Our signing protocol differs from many others in that it does *not* require the adversary's contributions to the secret key to be extracted during key generation, because the secret key shares are rerandomized and extracted afresh by $\mathcal{F}_{\mathsf{RVOLE}}$ for each signature. This liberates us in our choice of a key generation functionality: we need only ensure that the public key is uniformly sampled, and that the honest parties' shares are distributed consistently with the corrupt parties' shares. In this section, we achieve these guarantees *without* extracting the corrupt parties' shares; that is, our key generation protocol has *no* proofs of knowledge.

The intuition behind our relaxed functionality is simple: the ideal adversary (i.e. the simulator) sends it the corrupt parties' additive contributions to the *honest* parties' Shamir shares and to the public key, but the corrupt parties' additive contributions to their own Shamir shares are supplied only in the same group $\mathbb{G}$ as the public key (from which they cannot be recovered, under the hardness of discrete logarithm). The functionality can check (in $\mathbb{G}$) that these contributions form a polynomial of the correct degree, and if so, it samples a uniform additive contribution (i.e. another polynomial of correct degree) to the Shamir shares of all parties on behalf of the honest parties. As in other approaches, the public key is uniform from the perspective of the adversary, and the joint secret key is unrecoverable by the adversary (assuming that the discrete logarithm problem is hard in $\mathbb{G}$), so long as the number of corruptions is less than the threshold. Unlike other approaches, the secret key is *also* unrecoverable to the functionality itself, if the number of *honest* parties is less than the threshold.

We stress that our relaxed key generation functionality is sufficient for use with the signing protocol proposed in this work, but it *cannot* necessarily be

[10] See, for example, [Lin17, DKLs18, LN18, GG18, DKLs19, CGG+20, CCL+20, Lin22, HLNR23, CCL+23]. [DOK+20, ANO+22] use full-blown MPC for key generation.

used with other threshold signing protocols that use discrete-log keypairs. Consider for example, the classic folkloric threshold Schnorr protocol, as recently discussed by Lindell [Lin22]: in this protocol, the secret key shares of the corrupt parties are required in order to simulate signing, but they are *only* extracted during key generation. Consequently Lindell's protocol becomes unsimulatable if his key generation functionality is replaced by ours.

Finally, we note that our functionality is strictly weaker than the more typical functionality of Doerner et al. [DKL+23], which we discussed above. Because our functionality can be replaced with theirs, our protocol can also be replaced with theirs, if desired.

7.1 The Protocol

In this work, we have restricted ourselves to point-to-point authenticated channels and studiously avoided the use of broadcast channels. As a consequence we have achieved only security with selective abort. Because public keys may be linked *irrevocably* to identity or authority, it is often desirable to achieve at least *unanimous* abort in key generation protocols, so that each honest party is convinced that *all* honest parties can participate in signing and that a single agreed-upon public key exists, if no abort is observed. We present our key generation protocol in the hybrid model of a broadcast commitment functionality $\mathcal{F}_{\mathsf{Com}}(n)$, which behaves like the version of $\mathcal{F}_{\mathsf{Com}}$ introduced in section 3.1, except that it delivers consistent outputs to $n-1$ recipients simultaneously. This functionality is easily realized by combining any UC-secure commitment scheme with a broadcast channel. We will refer to the non-broadcast commitment functionality as $\mathcal{F}_{\mathsf{Com}}(2)$ hereafter.

When only point-to-point channels are assumed, as is otherwise the case in our work, our the broadcast channel underlying our enhanced $\mathcal{F}_{\mathsf{Com}}(n)$ can be replaced by the *echo-broadcast* technique of Goldwasser and Lindell [GL05], in which each broadcasted message is sent to all parties in a point-to-point fashion, and then the parties swap hashes of the broadcast channel to ensure that the messages they received are consistent. The result is that the adversary can force $\mathcal{F}_{\mathsf{Com}}(n)$ (and therefore our key generation protocol) to abort selectively. The echo can occur simultaneously with the last round of our key generation protocol, and thus the number of rounds is not affected by this technique.

On the other hand, if a true broadcast channel is available, then the final round of messages in our key generation protocol can also make use of it, and the result is that our protocol achieves unanimous abort.

Protocol 7.1. $\pi_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$**: Relaxed DLog Keygen**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ participate in this protocol and interact with the ideal functionality $\mathcal{F}_{\mathsf{Com}}$.

Key Generation:

1. On receiving $(\texttt{keygen}, \mathsf{sid})$ from the environment, each party $\mathcal{P}_i$ samples a random degree polynomial p_i of degree $t-1$ over $\mathbb{Z}_q$. Let P_i denote the corresponding polynomial in $\mathbb{G}$; i.e. let $P_i(x) = p_i(x) \cdot G$ for $x \in \mathbb{Z}_q$.

2. $\mathcal{P}_i$ computes $\mathbf{P}^{\text{-}j} := [n] \setminus \{j\}$ for every $j \in \mathbf{P}$, and then sends

   - $(\texttt{commit}, \mathcal{P}_i \| \mathcal{P}_{\mathbf{P}_1^{\text{-}i}} \| \ldots \| \mathcal{P}_{\mathbf{P}_{n-1}^{\text{-}i}} \| \mathsf{sid}, \{P_i(j)\}_{j \in [0,t-1]})$ to $\mathcal{F}_{\mathsf{Com}}(n)$.
   - $(\texttt{commit}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid}, p_i(j))$ to $\mathcal{F}_{\mathsf{Com}}(2)$ for $j \in \mathbf{P}^{\text{-}i}$.

3. Upon being notified of all other parties' commitments, each party $\mathcal{P}_i$ releases its committed values by sending

   - $(\texttt{decommit}, \mathcal{P}_i \| \mathcal{P}_{\mathbf{P}_1^{\text{-}i}} \| \ldots \| \mathcal{P}_{\mathbf{P}_{n-1}^{\text{-}i}} \| \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Com}}(n)$.
   - $(\texttt{decommit}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Com}}(2)$ for $j \in \mathbf{P}^{\text{-}i}$.

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

   - $(\texttt{opening}, \mathcal{P}_j \| \mathcal{P}_{\mathbf{P}_1^{\text{-}j}} \| \ldots \| \mathcal{P}_{\mathbf{P}_{n-1}^{\text{-}j}}, \{P_i(j)\}_{j \in [0,t-1]})$ from $\mathcal{F}_{\mathsf{Com}}(n)$
   - $(\texttt{opening}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid}, p_j(i))$ from $\mathcal{F}_{\mathsf{Com}}(2)$ for $j \in \mathbf{P}^{\text{-}i}$

   for each $j \in \mathbf{P}^{\text{-}i}$. Let P denote the sum of P_j for $j \in [n]$; i.e. let $P(x) = \sum_{j \in [n]} P_j(x)$ for $x \in \mathbb{Z}_q$. $\mathcal{P}_i$ computes

   $$P'(i) := \sum_{j \in [n]} p_j(i) \cdot G$$

   and then verifies that

   $$P'(i) = \begin{cases} P(i) & \text{if } i \in [t-1] \\ P(0) - \sum\limits_{j \in [t-1]} \mathsf{lagrange}([t-1] \cup \{i\}, j, 0) \cdot P(j) & \\ \hline \mathsf{lagrange}([t-1] \cup \{i\}, i, 0) & \text{otherwise} \end{cases}$$

   and if this equality holds, then $\mathcal{P}_i$ sends $(\texttt{ok}, \mathsf{sid})$ to all other parties. If this equality does not hold, then $\mathcal{P}_i$ sends $(\texttt{abort}, \mathsf{sid})$ to all parties.

5. If any party sends $(\texttt{abort}, \mathsf{sid})$ to $\mathcal{P}_i$, or $\mathcal{P}_i$ itself sent such a message, then $\mathcal{P}_i$ outputs $(\texttt{abort}, \mathsf{sid})$ to the environment, and performs no future instructions related to this session. If $\mathcal{P}_i$ transmitted $(\texttt{ok}, \mathsf{sid})$ and receives an identical message from all other parties, then it outputs $(\texttt{key-pair}, \mathsf{sid}, P(0), p(i))$ to the environment.

7.2 Proof of Security

Theorem 7.2 (Key Generation Security Theorem)**.** *For every group described by $\mathcal{G}$ and every malicious adversary $\mathcal{A}$ that statically corrupts up to $t-1$ parties, there exists a simulator $\mathcal{S}^{\mathcal{A}}_{\mathsf{RelaxedKeyGen}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$ it holds that*

$$\left\{\text{REAL}_{\pi_{\mathsf{RelaxedKeyGen}}(\mathcal{G},n,t),\mathcal{A},\mathcal{Z}}\left(\lambda_{\mathsf{s}},z\right)\right\}_{\lambda_{\mathsf{s}}\in\mathbb{N},n\in\mathbb{N}:n>1,t\in[2,n],z\in\{0,1\}^*}$$
$$=\left\{\text{IDEAL}_{\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G},n,t),\mathcal{S}^{\mathcal{A}}_{\mathsf{RelaxedKeyGen}}(\mathcal{G},n,t),\mathcal{Z}}\left(\lambda_{\mathsf{s}},z\right)\right\}_{\lambda_{\mathsf{s}}\in\mathbb{N},n\in\mathbb{N}:n>1,t\in[2,n],z\in\{0,1\}^*}$$

Proof. This proof is direct, without any hybrid experiments. The simulator is as follows:

Simulator 7.3. $\mathcal{S}^{\mathcal{A}}_{\mathsf{RelaxedKeyGen}}(\mathcal{G},n,t)$**: Relaxed DLog Keygen**

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

Key Generation:

1. On receiving $(\texttt{keygen-req},\mathsf{sid},j)$ from $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G},n,t)$ such that $j\in[n]\setminus\mathbf{P}^*$, compute $\mathbf{P}^{-i}:=[n]\setminus\{i\}$ for every $i\in\mathbf{P}$ and send to each $\mathcal{P}_i$ for $i\in\mathbf{P}^*$
   - $(\texttt{committed},\mathcal{P}_j\|\mathcal{P}_{\mathbf{P}^{-j}_1}\|\ldots\|\mathcal{P}_{\mathbf{P}^{-j}_{n-1}}\|\mathsf{sid}))$ on behalf of $\mathcal{F}_{\mathsf{Com}}(n)$.
   - $(\texttt{committed},\mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}(2)$.
2. On receiving
   - $(\texttt{commit},\mathcal{P}_i\|\mathcal{P}_{\mathbf{P}^{-i}_1}\|\ldots\|\mathcal{P}_{\mathbf{P}^{-i}_{n-1}}\|\mathsf{sid},\{P_i(j)\}_{j\in[0,t-1]})$ on behalf of $\mathcal{F}_{\mathsf{Com}}(n)$.
   - $(\texttt{commit},\mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid},p_i(j))$ on behalf of $\mathcal{F}_{\mathsf{Com}}(2)$ for some *consistent* $j\in\mathbf{P}^{-i}\setminus\mathbf{P}^*$.

   from every $\mathcal{P}_i$ for $i\in\mathbf{P}^*$
   - If there exists some $j'\in\mathbf{P}^{-i}\setminus\mathbf{P}^*$ with respect to which such messages have not all been received, such that $j'\neq j$, then sample $p_j(i)\leftarrow\mathbb{Z}_q$

uniformly for each $i \in \mathbf{P}^*$ and sample P_j as a uniform polynomial of degree $t-1$ over $\mathbb{G}$ such that $P_j(i) = p_j(i) \cdot G$ for every $i \in \mathbf{P}^*$.

- If j is the *last* value which satisfies the above conditions, then
  (a) For every $i \in \mathbf{P}^*$ and $k \in [t, n]$ compute

$$P_i(k) := \frac{P_i(0) - \sum_{l \in [t-1]} \mathsf{lagrange}([t-1] \cup \{k\}, l, 0) \cdot P_i(l)}{\mathsf{lagrange}([t-1] \cup \{k\}, k, 0)}$$

  and then for every $k \in [n] \setminus \mathbf{P}^*$ compute

$$\check{p}(k) := \sum_{i \in \mathbf{P}^*} p_i(k)$$
$$\check{P}(k) := \sum_{i \in \mathbf{P}^*} P_i(k)$$

  (b) Send $(\texttt{adv-poly}, \mathsf{sid}, \{\check{p}(k)\}_{k \in [n] \setminus \mathbf{P}^*}, \{\check{P}(i)\}_{i \in \mathbf{P}^*})$ directly to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.
  (c) On receiving $(\texttt{hon-poly}, \mathsf{sid}, P(0), \{\hat{P}(k)\}_{k \in [n] \setminus \mathbf{P}^*}, \{\hat{p}(i)\}_{i \in \mathbf{P}^*})$ from $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, compute

$$\hat{P}(i) := \hat{p}(i) \cdot G \quad \text{for } i \in \mathbf{P}^*$$
$$p_j(i) := \hat{p}(i) - \sum_{k \in \mathbf{P}^{\text{-}j} \setminus \mathbf{P}^*} p_k(i) \quad \text{for } i \in \mathbf{P}^*$$
$$P_j(k) := \hat{P}(k) - \sum_{i \in \mathbf{P}^{\text{-}j} \setminus \mathbf{P}^*} P_j(k) \quad \text{for } k \in [t-1]$$
$$P_j(0) := P(0) - \sum_{i \in \mathbf{P}^{\text{-}j}} P_i(0)$$

  (d) If $(\texttt{abort}, \mathsf{sid})$ is received from $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$ instead of $\texttt{hon-poly}$, then sample $p_j(i) \leftarrow \mathbb{Z}_q$ uniformly for each $i \in \mathbf{P}^*$ and sample P_j as a uniform polynomial of degree $t-1$ over $\mathbb{G}$ such that $P_j(i) = p_j(i) \cdot G$ for every $i \in \mathbf{P}^*$.

  and finally, send to each $\mathcal{P}_i$ for $i \in \mathbf{P}^*$

- $(\texttt{opening}, \mathcal{P}_j \| \mathcal{P}_{\mathbf{P}_1^{\text{-}j}} \| \ldots \| \mathcal{P}_{\mathbf{P}_{n-1}^{\text{-}j}} \| \mathsf{sid}, \{P_j(k)\}_{k \in [0, t-1]})$ on behalf of $\mathcal{F}_{\mathsf{Com}}(n)$.
- $(\texttt{opening}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid}, p_j(i))$ on behalf of $\mathcal{F}_{\mathsf{Com}}(2)$.

3. On receiving

- $(\texttt{decommit}, \mathcal{P}_i \| \mathcal{P}_{\mathbf{P}_1^{\text{-}i}} \| \ldots \| \mathcal{P}_{\mathbf{P}_{n-1}^{\text{-}i}} \| \mathsf{sid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}(n)$.

- $(\texttt{decommit}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid})$ on behalf of $\mathcal{F}_{\mathsf{Com}}(2)$ for some *consistent* $j \in \mathbf{P}^{-i} \setminus \mathbf{P}^*$.

from every $\mathcal{P}_i$ for $i \in \mathbf{P}^*$, if $\check{p}(j) \cdot G \neq \check{P}(j)$, then send $(\texttt{abort}, \mathsf{sid})$ to all corrupt parties on behalf of $\mathcal{P}_j$ and send $(\texttt{abort}, \mathsf{sid}, k)$ to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$ for every $k \in [n] \setminus \mathbf{P}^*$. Otherwise send $(\texttt{ok}, \mathsf{sid})$ to all corrupt parties on behalf of $\mathcal{P}_j$.

4. On receiving $(\texttt{abort}, \mathsf{sid})$ from $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ for some $i \in \mathbf{P}^*$ and $j \in [n] \setminus \mathbf{P}^*$, send $(\texttt{abort}, \mathsf{sid}, j)$ to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.
5. On receiving $(\texttt{ok}, \mathsf{sid})$ from $\mathcal{P}_i$ on behalf of $\mathcal{P}_j$ some consistent $j \in [n] \setminus \mathbf{P}^*$ and *every* $i \in \mathbf{P}^*$, if $\check{p}(k) \cdot G = \check{P}(k)$ for every $k \in [n] \setminus \mathbf{P}^*$, then send $(\texttt{release}, \mathsf{sid}, j)$ to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.

The view of the adversary is completely characterized by the values transmitted by the corrupt parties and by the values emitted by the honest parties, both in the protocol and to the environment. That is, the adversary's view is characterized by the public key $P(0)$ and for every $j \in [n] \setminus \mathbf{P}^*$ by the secret key share $p(j)$, the polynomial P_j, and the selected discrete logarithms $p_j(i)$ of points on P_j, for $i \in \mathbf{P}^*$.

We argue first about the distribution of aborts, then about the distribution of values when an abort occurs, then about the distribution of values when an abort does not occur.

We observe first of all that in the ideal world, the values $\{\check{P}(i)\}_{i \in \mathbf{P}^*}$ supplied to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$ by $\mathcal{S}_{\mathsf{RelaxedKeyGen}}$ are interpolated from the adversary's commitments to t polynomial points. These, therefore, lie on a degree $t-1$ polynomial (or many such polynomials, if $|\mathbf{P}^*| < t-1$), and it follows that an abort based on the conditions in step 1 only occurs if the values $\{\check{p}(k)\}_{k \in [n] \setminus \mathbf{P}^*}$ are inconsistent with this polynomial (or with all polynomials that pass through $\{\check{P}(i)\}_{i \in \mathbf{P}^*}$, if $|\mathbf{P}^*| < t-1$). Recall that per step 2,

$$\check{p}(k) = \sum_{i \in \mathbf{P}^*} p_i(k) \qquad \text{and} \qquad \check{P}(k) = \sum_{i \in \mathbf{P}^*} P_i(k)$$

for every $k \in [n] \setminus \mathbf{P}^*$. In the real world, per step 4 of $\pi_{\mathsf{RelaxedKeyGen}}$, each honest party $\mathcal{P}_k$ for $k \in [n] \setminus \mathbf{P}^*$ signals the other parties to abort if

$$\sum_{i \in [n]} p_i(k) \cdot G \neq \sum_{i \in [n]} P_i(k)$$

where $P_i(k)$ is interpolated from the first t points on P_i, if necessary. Since $p_j(k) \cdot G = P_j(k)$ for all $j, k \in [n] \setminus \mathbf{P}^*$ by construction, this condition is equivalent to the ideal-world abort condition expressed by $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$.

Next, we analyze the distributions of the values in the protocol, conditioned on an abort occurring. In the ideal world, the $\mathcal{S}_{\mathsf{RelaxedKeyGen}}$ samples p_k for $k \in [n] \setminus \mathbf{P}^*$ uniformly, whereas in the real world, $\mathcal{P}_k$ samples p_k for $k \in [n] \setminus$

$\mathbf{P}^*$ uniformly. In both worlds, we have $P_k(x) = p_k(x) \cdot G$ for $x \in \mathbb{Z}_q$, and neither world does the environment learn $p(k)$, due to the fact that an abort occurs. Thus the two worlds are identically distributed, conditioned on an abort occurring.

If an abort does not occur, then the situation is very similar. The distinction is that $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$ samples $\hat{p}$ uniformly, defines $\hat{P}(x) = \hat{p}(x) \cdot G$ for $x \in \mathbb{Z}_q$, and sends $\{\hat{P}(k)\}_{k \in [n] \setminus \mathbf{P}^*}$ and $\{\hat{p}(i)\}_{i \in \mathbf{P}^*}$ to $\mathcal{S}_{\mathsf{RelaxedKeyGen}}$. If we let $\mathcal{P}_h$ denote the *last* honest party to decommit in step 4 of the protocol, then we can see that in the ideal world, $\mathcal{S}_{\mathsf{RelaxedKeyGen}}$ samples p_k for $k \in \mathbf{P}^{\text{-}h} \setminus \mathbf{P}^*$ uniformly as before, calculates $P_h(x)$ such that

$$\hat{P}(x) = \sum_{j \in [n] \setminus \mathbf{P}^*} P_j(x)$$

for every $x \in \mathbb{Z}_q$, and calculates $p_h(i)$ such that

$$\hat{p}(i) = \sum_{j \in [n] \setminus \mathbf{P}^*} p_j(i)$$

for $i \in \mathbf{P}^*$. Thus the joint distributions of P_j for $j \in [n]$ and $p_j(i)$ for $j \in [n]$ and $i \in \mathbf{P}^*$ are identical in the real and ideal worlds. It remains finally to observe that in the ideal world, each honest party $\mathcal{P}_j$ emits $p(j) = \check{p}(j) + \hat{p}(j)$, whereas in the real world, it emits

$$p(i) = \sum_{j \in [n]} p_j(i)$$

which is distributed identically. □

8 Analytical Efficiency

In this section, we give a closed-form accounting of the bandwidth costs of our protocol and its various building blocks. We account for the number of elliptic curve scalar operations that our protocol requires during signing, since this is a substantial portion of the computational cost. We begin by analyzing the realizations for non-VOLE building blocks that were suggested in section 3.1: an $\mathcal{F}_{\mathsf{Com}}$ commitment to any payload requires $2\lambda_{\mathsf{c}}$ bits to be transmitted, and a decommitment to a payload of size x requires $2\lambda_{\mathsf{c}} + x$ bits. We assume that the "broadcast" variant of $\mathcal{F}_{\mathsf{Com}}$ multiplies the foregoing costs by the number of recipients, and that then adds $2\lambda_{\mathsf{c}}$ transmitted bits per recipient in order to implement echo-broadcast. We assume $\mathcal{F}_{\mathsf{Zero}}(\mathbb{Z}_q, t)$ requires a one-time setup cost of $(t-1)$ commitments and decommitments to λ_{c} bits on the part of each party, and that invocation by that same set of parties is free in terms of bandwidth thereafter.

8.1 Oblivious Transfer

Our VOLE protocol relies on OT, which we realize via the OT-extension protocol of Roy [Roy22], using the one-round optimization introduced in section 5.1, with

base OTs supplied by the two-round UC-secure endemic OT protocol of Masny and Rindal [MR19]. We instantiate the latter primitive from the decisional Diffie-Hellman assumption over the same group $\mathcal{G}$ in which signatures are to be computed. For ℓ_{OT} OT instances, the protocol of Masny and Rindal has an average per-party bandwidth cost of

$$\mathsf{EOTCost}(|G|, \ell_{\mathsf{OT}}) \mapsto 2 \cdot |G| \cdot \ell_{\mathsf{OT}}$$

and it requires each party to compute $3\ell_{\mathsf{OT}}$ elliptic curve scalar operations, on average.

Roy's protocol requires a one-time setup that comprises exactly λ_{c} instances of OT. This is followed by any number of extension batches. Per the accounting of Doerner et al. [DKL+23], each batch of ℓ_{OTE} endemic OT instances has an average per-party bandwidth cost of

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

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

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

where $|m|$ is the size of the correlation in bits.

8.2 Our VOLE

For the purposes of our cost analysis, we assume that setup for endemic OT extension is performed once per pair of parties, and reused thereafter in all instances of the protocol realizing $\mathcal{F}_{\mathsf{EOTE}}$ among those two parties. Our protocol involves an endemic OT-extention batch of size $\ell_{\mathsf{OTE}} = \xi = \kappa + 2\lambda_{\mathsf{s}}$, and the transmission of $(\ell + 1) \cdot \xi + 1$ elements of $\mathbb{Z}_q$ and $2\lambda_{\mathsf{c}}$ bits directly from Alice to Bob. This brings the total online (i.e. excluding one-time setup) average per-party bandwidth cost of our DKLs-derived VOLE protocol to

$$\begin{aligned}&\mathsf{VOLECost}(\lambda_{\mathsf{c}}, \lambda_{\mathsf{s}}, \kappa, \ell) \mapsto \\ &\quad \mathsf{EOTECost}(\lambda_{\mathsf{c}}, \kappa + 2\lambda_{\mathsf{s}}) + (\kappa/2 + \lambda_{\mathsf{s}}) \cdot (\ell + 1) \cdot \kappa + \kappa/2 + \lambda_{\mathsf{c}}\end{aligned}$$

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

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

8.3 VOLE from HMRT22

Next, we present the cost of using an alternate VOLE protocol derived from the work of Haitner et al. [HMRT22]. Their protocol realizes a weaker functionality than the one we have specified, and we have *not* proven the combination secure, and we remind the reader that the protocol of Haitner et al. requires an additional round, relative to the DKLs-derived VOLE described in section 5. Nevertheless, we include a cost analysis of their protocol for the sake of comparison.

Haitner et al.'s description of their protocol specifies correlated OT rather than correlated OT-extension. In order to make an apples-to-apples comparison against our VOLE, we assume OT-extension is used, and as a consequence it is necessary to perform a one-time setup procedure. Thus

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

The evaluation stage of their protocol involves a batch of $\ell_{\mathsf{OTE}} = \kappa + 4\lambda_\mathsf{s}$ correlated OT instances. Per a random-oracle-based optimization mentioned in their paper, the only additional data that must be sent is a single λ_c-bit seed, plus a single element of $\mathbb{Z}_q$.[11] If we derive a VOLE from their OLE in the same way that we derived π_{RVOLE} from the DKLs OLE protocols, then the average per-party bandwidth cost is

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

Assuming that SoftSpokenOT is used to realize the correlated OT-extension, their protocol outperforms ours in terms of bandwidth costs only when

$$\frac{2\lambda_\mathsf{c} \cdot \lambda_\mathsf{s}}{k_{\mathsf{SSOT}}} + 2\lambda_\mathsf{s} \cdot \kappa \cdot \ell + \kappa \cdot \ell < \kappa^2 + 2\lambda_\mathsf{s} \cdot \kappa + \kappa + \lambda_\mathsf{c}$$

8.4 Our Key Generation and ECDSA Protocols

When the echo-broadcast synchronization messages are coalesced, the bandwidth cost of realizing $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$ via the protocol introduced in section 7 is given by

$$\mathsf{KeyGenCost}(n, \lambda_\mathsf{c}, \kappa, |G|) \mapsto (n-1) \cdot (10\lambda_\mathsf{c} + t \cdot |G| + \kappa)$$

The one-time initialization for π_{ECDSA} involves running $\mathcal{F}_{\mathsf{RelaxedKeyGen}}$ and initializing two instances of $\mathcal{F}_{\mathsf{RVOLE}}$ per pair of parties. Each party must also commit and release a pair of λ_c-bit seeds, in order to initialize the protocol that realizes $\mathcal{F}_{\mathsf{Zero}}$. Our signing protocol is very simple. Each pair of parties performs two VOLE evaluations, and each party $\mathcal{P}_i$ commits and releases R_i

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

and transmits $\mathbf{\Gamma}^{\mathsf{u}}_{i,j}$, $\mathbf{\Gamma}^{\mathsf{v}}_{i,j}$, $\boldsymbol{\psi}_{i,j}$, pk_i, w_i, and u_i to every $\mathcal{P}_j$ such that $j \neq i$. This gives us a total average per-party bandwidth cost of

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

Finally, each party must perform $6t - 2$ elliptic curve scalar operations in order to generate a signature. To set up their VOLE instances (for all counterparties), each party must perform $6\lambda_{\mathsf{c}} \cdot (n-1)$ scalar operations. To generate a shared keypair, each party must perform at most $2t$ scalar operations.

8.5 Concrete Results

In table 1, we substitute values into the above equations to derive the concrete average per-party bandwidth costs for common security parameters. We assume that point compression is used for elements of $\mathbb{G}$, such that they require only one byte more than elements of $\mathbb{Z}_q$.[12] In all cases, we assume that $\kappa = 2\lambda_{\mathsf{c}}$ and $\lambda_{\mathsf{s}} = 80$. Note that the seeds used to initialize the VOLE protocol and the protocol that realized $\mathcal{F}_{\mathsf{Zero}}$ can be combined, which implies that the cost of VOLE setup (and therefore the overall setup cost) is the same regardless of which VOLE method is used.

For comparison, when $\lambda_{\mathsf{c}} = 256$ and $\lambda_{\mathsf{s}} = 80$, the 2-of-$n$ signing protocol of Doerner et al. [DKLs18] requires each party to send 116.4 KiB (on average), whereas our new protocol (with our new DKLs-derivedVOLE) requires only 49.7 KiB per party to be sent. On the other hand, our new protocol has the same communication pattern as theirs (under pipelining), requires fewer elliptic curve scalar operations than theirs does,[13] realizes a standard functionality, whereas their functionality allows the adversary to bias R, and achieves statistical security, whereas theirs is secure only assuming that the computational Diffie-Dellman problem is hard in $\mathbb{G}$ and that ECDSA is a signature scheme over $\mathcal{G}$. We therefore claim that our protocol is strictly superior to the original 2-of-n DKLs protocol.

The t-of-n protocol of Doerner et al. [DKLs19] requires $(t-1) \cdot 88.3$ KiB to be sent by each party (on average) when the $(\lceil \log_2(t) \rceil + 6)$-round variant is used.[14] Our new protocol requires only $(t-1) \cdot 49.7$ KiB to be sent, has only three rounds (or two, if pipelining is employed), and achieves statistical security if the VOLE is ideal, whereas theirs is secure assuming that the computational Diffie-Hellman problem is hard in $\mathbb{G}$. However, their protocol requires each party to compute only 6 elliptic curve scalar operations during signing, and ours requires $6t - 2$. Given the efficiency of elliptic curve operations on modern hardware, and the additional latency incurred by additional parties, we believe

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

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

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

our new protocol to have the advantage in nearly any real-world deployment scenario.

λ_c	128	192	256
κ	256	384	512
$\lvert G\rvert$	264	392	520
Setup	$(n-1)\cdot 137232$	$(n-1)\cdot 304144$	$(n-1)\cdot 536592$
Signing (our VOLE)	$(t-1)\cdot 406752$	$(t-1)\cdot 812864$	$(t-1)\cdot 1354144$
Signing (HMRT22)	$(t-1)\cdot 392544$	$(t-1)\cdot 742400$	$(t-1)\cdot 1194656$

Table 1: Bandwidth Costs, in total bits transmitted per party, for t signers out of n total parties. We assume the worst-case cost for setup; i.e. $t = n$. Note that in all cases, the statistical parameter $\lambda_s = 80$.

9 A Two-Round Protocol for Honest Majorities

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

Theorem 9.1 (Informal Honest-Majority Security Theorem)**.** *When* $(t \text{ choose } \lceil t/2 \rceil) \in \text{poly}(\lambda)$*, there exists a two-round protocol that UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *against a malicious adversary that statically corrupts fewer than* $t/2$ *parties, assuming the existence of pseudo-random functions.*

The protocol begins by sampling *replicated* secret shares of sk, r, $\zeta = 0$, and ϕ, with a reconstruction threshold of $\lceil t/2 \rceil$. To non-interactively generate replicated secret shares of zero, there is a direct extension of the protocol we have given for realizing $\mathcal{F}_{\mathsf{Zero}}$ in section 3.1. To non-interactively sample replicated secret shares of a *uniform* value, one can use a classic protocol of Cramer, Damgård, and Ishai [CDI05]: simply replicate shares of a seed, and use a PRF to expand the replicated seeds when necessary.

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

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

Honest-majority techniques make our consistency check and the commit-and-release mechanism for R superfluous. Since the honest parties' shares fully specify R, any attempt by the adversary to bias this value will result in a polynomial of incorrect degree, which can be detected. Since the multiplication operations are completely non-interactive, any cheating on the part of the adversary *must* be independent of the honest parties' secrets, and therefore expressable in terms of simple linear offsets relative to the expected values, which can be perfectly detected by verifying the signature, just as in the protocol from section 3. In terms of communication, each party must only send a single share of R to the others, followed by one share each of u and w; thus, when $\kappa = 256$, the total amount of data sent by every party to each of the others is 776 bits.

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

Acknowledgements

The authors of this work are supported by NSF grants 1646671, 1816028, and 2055568, by the ERC projects NTSC (742754), SPEC (803096), and HSS (852952), by ISF grant 2774/2, by AFOSR award FA9550-21-1-0046, by the Azrieli Foundation, by the Brown University Data Science Institute, and by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

References

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *Advances in Cryptology – CRYPTO 2017, part I*, 2017.

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

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

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology – CRYPTO 2019, part III*, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS)*, 2018.

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

[BDM22] Pedro Branco, Nico Döttling, and Paulo Mateus. Two-round oblivious linear evaluation from learning with errors. In *Proceedings of the 25th International Conference on the Theory and Practice of Public-Key Cryptography (PKC), part I*, 2022.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology – EUROCRYPT 2011*, 2011.

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

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

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology – ASIACRYPT 2001*, 2001.

[BP23] Luís T. A. N. Brandão and René Peralta. NISTIR 8214c ipd NIST first call for multi-party threshold schemes (initial public draft). In *Computer Security Resource Center*, 2023.

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

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

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *Proceedings of the 23rd International Conference on the Theory and Practice of Public-Key Cryptography (PKC), part II*, 2020.

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

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Proceedings of the 2nd Theory of Cryptography Conference (TCC)*, 2005.

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

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. 2021.

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

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

[CMI93] Manuel Cerecedo, Tsutomu Matsumoto, and Hideki Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. In *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E76-A, No.4, pp.532-545*, 1993.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *Advances in Cryptology – CRYPTO 2021, part III*, 2021.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Blazing fast OT for three-round UC OT extension. In *Proceedings of the 23rd International Conference on the Theory and Practice of Public-Key Cryptography (PKC), part II*, 2020.

[DJN+20] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. 2020.

[dJV21] Leo de Castro, Chiraag Juvekar, and Vinod Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, 2021.

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

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

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

[DKLs24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA in three rounds. In *Proceedings of the 45th IEEE Symposium on Security and Privacy (S&P)*, 2024.

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *Proceedings of the 25th European Symposium on Research in Computer Security (ESORICS), Part II*, 2020.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, 2012.

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

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

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *Advances in Cryptology – CRYPTO 2018, part II*, 2018.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS)*, 2018.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. 2016.

[Gil99] Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology – CRYPTO 1999*, 1999.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *Advances in Cryptology – EUROCRYPT 1996*, 1996.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3), 2005.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, 1987.

[GS22a] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506, 2022.

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

[HLNR23] Iftach Haitner, Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Paper 2018/987, Version 20230529:135032, 2023.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold paillier in the two-party setting. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA)*, 2012.

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

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO 2003*, 2003.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4), 1996.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology – CRYPTO 2015, part I*, 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 23th ACM Conference on Computer and Communications Security (CCS)*, 2016.

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

[Lan95] Susan K. Langford. Threshold dss signatures without a trusted party. In *Advances in Cryptology – CRYPTO 1995*, 1995.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In *Advances in Cryptology – CRYPTO 2017, part II*, 2017.

[Lin21] Yehuda Lindell. Secure multiparty computation. *Communications of the ACM*, 64(1), 2021.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Paper 2022/374, 2022.

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

[MR01] Philip D. MacKenzie and Michael K. Reiter. Two-party generation of DSA signatures. In *Advances in Cryptology – CRYPTO 2001*, 2001.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, 2019.

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

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

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

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In *Advances in Cryptology – CRYPTO 2003*, 2003.

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

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

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

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, 2019.

[ST19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In *IMA International Conference on Cryptography and Coding*, 2019.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *Proceedings of the 27th ACM Conference on Computer and Communications Security (CCS)*, 2020.