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

2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00178

***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. [2] with an efficient statistical consistency check reminiscent of the ones used by the protocols of Doerner et al. [3], [4]. 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. We demonstrate empirically that our protocol outperforms those of Doerner et al. by factors of as much as six in high-latency environments, and that it is multiple orders of magnitude faster than Paillier-based approaches.**

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) [5], 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 [6], 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 [7].

Threshold signing schemes are straightforward to construct for some elliptic curve signatures, such as BLS [8] and Schnorr [9], 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 $b = r^{\times}$ are effectively public coefficients, $r^{\times}$ 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 [10] 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

The full version [1] is available: https://eprint.iacr.org/2023/765

© 2024, Jack Doerner. Under license to IEEE.

DOI 10.1109/SP54263.2024.00178
Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

by several prior works [11], [2], [12]. 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. [3], [4]. 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. [3], [4] 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 [13], [14], and two rounds fewer than the best-known protocol from general assumptions (i.e. OT) [15]. It also brings the round complexity of threshold ECDSA to par with threshold Schnorr [16] 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 [19] 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 $t > n/2$ [20]. 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 [10]. 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. [20] and Cerecedo et al. [21] used *verifiable secret sharing* (VSS) techniques to build *robust* protocols when $n \geq 3t + 1$, and Damgård et al. [22] constructed a simple protocol that is secure with

1. A number of two-round distributed Schnorr protocols such as MuSig2 [17] and FROST [18] also exist, with game-based security under non-standard assumptions.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

abort against malicious adversaries when $n \geq 2t+1$. More recently, Groth and Shoup [12] developed the first distributed ECDSA protocol that achieves guaranteed output delivery in the *asynchronous* setting when $n \geq 3t+1$, 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 [23] 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 [24]. Fifteen years later, Gennaro et al. [25] used *threshold* AHE to extend this idea to the many-party setting, and then Boneh et al. [26] 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 [27], [28], [29], 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 [30] 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. [3] 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 [31] 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 [32], [33], [34] 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. [4] 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 [3], [4] 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 assumptions 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 [35] 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 [25], [26] 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. [20], 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 [11] 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 [36], which induces an overhead of roughly one order of magnitude in terms of computation and communication.

Somewhat later, Canetti et al. [13] presented a four-round signing protocol that essentially followed the same

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

core protocol layout as Gennaro and Goldfeder [35], but they replaced the interactive masked signature verification to validate honest behaviour with a conceptuatlly-straightforward GMW-style [37] 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. [15] 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. [38] started from the protocol of Canetti et al. and replaced Paillier encryption with the Castagnos-Laguillaumie (CL) encryption scheme [39] 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. Wong et al. [40] showed how the protocol of Castagnos et al. could be made flexible to accommodate a variation in participants across signing rounds.

Generic Approaches. Smart and Talibi [41] and Dalskov et al. [42] concurrently proposed generic MPC approaches to ECDSA signing. Both groups observed that the MAC checks of SPDZ-style [43] 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 [42]: 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. [2] 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 [11] 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 [44], 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 [11] and later explicitly construed as a random correlation by Abram et al. [2]. 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. [2], 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. [3], [4] 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.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

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. [3] 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 [4]. In this work, we refine their techniques and propose a new VOLE protocol[2] that has lower concrete 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.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

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 [45], [46], it is possible to implement our protocol entirely from *native* assumptions, much as Doerner et al. did [3], [4]. 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 savings is paramount, then a VOLE derived from additively-homomorphic encryption [13], [38] 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 [36] or Kondi-shelat [47] 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 [3], [4], [42]. In terms of bandwidth, our combined protocol is competitive with techniques based on *weak* multiplication [48], [15], 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. [3], [4] have heretofore remained the state of the art due to the fact that competing approaches based upon Paillier encryption [24], [13] or class groups [39], [38] 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 in section 4 we discuss a proof-of-concept implementation and report benchmark results in a number of settings. In the appendix, we provide a proof of security for our signing protocol. In the full version of this paper [1], we give our random VOLE construction, we give a security proof for our random VOLE, we introduce a drastically optimized key generation protocol with *no proofs of knowledge* and prove it secure, we give a closed form cost analysis of all of our protocols, and we give a brief account of a significantly simplified two-round honest-majority protocol.

2. Preliminaries

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,

3. We introduce this functionality and the rationale behind it in the full version of this paper [1]. For now it can be thought of as as performing batches of oblivious transfers, with random inputs.

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

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

they are given indices A and B and referred to as Alice and Bob, respectively. The threshold is denoted t.

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 [49]. 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.

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 $\mathcal{G} \leftarrow \mathsf{GrpGen}(1^{\lambda_c})$ against which the discrete logarithm assumption must hold.[5] In practice, the group description is fixed and standardized.

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

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

Algorithm 2.2. $\mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$

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 $\sigma := (s, r^{\mathsf{x}})$.

3. Three-Round Threshold ECDSA

We present the functionality that our threshold ECDSA protocol realizes. In contrast to the functionality given by Doerner et al. [4], 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. $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$

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 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
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 $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$.
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 $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, i)$.
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.

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

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

We begin with a functionality that samples Shamir sharings of keys for discrete-log cryptosystems. 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 the full version [1] we discuss this design decision and its implications, and introduce a protocol to realize our functionality.

Functionality 3.2. $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$

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 $\mathcal{P}_i$.

We use the standard $\mathcal{F}_{\mathsf{Com}}$ commitment functionality of Canetti et al. [50], which can be realized in the random oracle model via the folklore method of simply hashing a (privately salted) message.

We use the $\mathcal{F}_{\mathsf{Zero}}$ functionality of Doerner et al. [51], which non-interactively samples uniform secret-sharings of zero. This is straightforward to realize with any PRF: each pair of parties agrees on a common PRF key during setup, and when given a PRF input online, one party adds the PRF output while the other subtracts it.

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. In the full version of this paper [1], we propose to realize $\mathcal{F}_{\mathsf{RVOLE}}$ via an optimized, two-round version of the OLE protocol of Doerner et al. [4], which in turn is a hardening of the OT-based OLE of Gilboa [31] for malicious security. We elaborate no further here.

Functionality 3.3. $\mathcal{F}_{\mathsf{RVOLE}}(q, \ell)$

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.

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

6. As we discuss in the full version [1], this is equivalent to plain VOLE under a simple information-theoretic transformation.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

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 3.4. $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$

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 $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.

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 $(\mathtt{key\text{-}pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ in memory. If not, then $\mathcal{P}_i$ sends $(\mathtt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$.
2) On receiving $(\mathtt{key\text{-}pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ from $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$ each $\mathcal{P}_i$ stores this message in memory and outputs $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ 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 $(\mathtt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ 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 $(\mathtt{key\text{-}pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ 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

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

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

   This completes the first round.

if pipelining, supply m here[b]

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

   $\mathcal{P}_i$ computes

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

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

$$\mathbf{\Gamma}^{\mathsf{u}}_{i,j} := \mathbf{c}^{\mathsf{u}}_{i,j} \cdot G \qquad \mathsf{pk}_i := \mathsf{sk}_i \cdot G$$

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

$$\boldsymbol{\Gamma}^{\mathsf{v}}_{i,j} \coloneqq \mathbf{c}^{\mathsf{v}}_{i,j} \cdot G \qquad \boldsymbol{\psi}_{i,j} \coloneqq \phi_i - \boldsymbol{\chi}_{i,j}$$

for every $j \in \mathbf{P}^{-i}$ and for every $j \in \mathbf{P}^{-i}$ 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 $\mathcal{P}_j$

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

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

   for every $j \in \mathbf{P}^{-i}$, and whether $\sum_{k \in \mathbf{P}} \mathsf{pk}_k = \mathsf{pk}$, and if these equations hold, then $\mathcal{P}_i$ computes

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

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

Pipelining. The first round of the protocol can be evaluated before the message is known. A set of parties signing multiple messages in sequence can preprocess the first round of the next instance in parallel with the current instance, at no harm to security, as no information about R is revealed in the first round.

4. Empirical Efficiency

We created a proof-of-concept implementation of our protocol in roughly ten thousand lines of Rust code including unit tests. Our implementation uses the `secp256k1` elliptic curve via the library of Doerner et al. [3], [4] and SHA-NI instructions for hardware-accellerated hashing on `x86_64` machines. Our implementation of SoftSpokenOT follows the simplified description of Keller et al. [33], with the two-round modification discussed in the full version of this paper. Our implementation is multithreaded, and in the interest of speed it does *not* use point compression when transmitting elliptic curve elements, which results in higher bandwidth use than strictly necessary. Our benchmarking programs establish insecure connections among the parties one time only, and then measure the wall-clock time of setup or signing operations. Thus, they measure the impact of latency, computation, and bandwidth constraints, but not overhead due to channel establishment or authentication. We count bytes sent and received, including session ID and data serialization overhead (but not authentication overhead, TCP overhead, etc.). We compiled our code against stable Rust 1.74 and set $\lambda_{\mathsf{c}} = 128$ and $\lambda_{\mathsf{s}} = 80$.

Benchmark Design. We performed benchmarks in three network settings, which we will refer to as *LAN*, *US WAN*, and *Global WAN*. In all settings, each party was allocated a single Google Cloud Platform (GCP) `n2d-highcpu-8` node, running Debian 11 with kernel 5.10.179, with a CPU from the AMD Epyc Milan family that has four virtual cores clocked at 2.45 GHz that can execute eight threads in total.

In the LAN setting, all parties were located in a single datacenter in South Carolina. Internal latency in this datacenter was measured at 0.4ms, and thus the protocol is essentially compute-bound in this setting. In the US WAN setting, the parties were divided equally among four datacenters in South Carolina, Nevada, Iowa, and Virginia. The longest observed latency among these four was 60ms between Virginia and Nevada. In the Global WAN setting, the parties were divided equally among sixteen global datacenters depicted in figure 1: the four aforementioned datacenters in the USA, plus datacenters in Montréal, São Paulo, London, the Netherlands, Zurich, Israel, Mumbai, Singapore, Taiwan, Seoul, Tokyo, and Sydney. The longest observed latency among these sixteen was 329ms between Mumbai and São Paulo. In the WAN contexts, our experiments always included an equal number of nodes in each region.

Because the parties are in different geographic regions and there is a single initiator, they begin the

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

protocol at slightly different times. In our benchmarks, all parties measure their locally-observed execution time from the moment they receive an *experiment begin* message from the initiator to the moment they produce an output. Reported measurements are averaged over all parties and at least 60 samples. Wall-clock times are reported in Table 1 and bytes transmitted in Table 2.

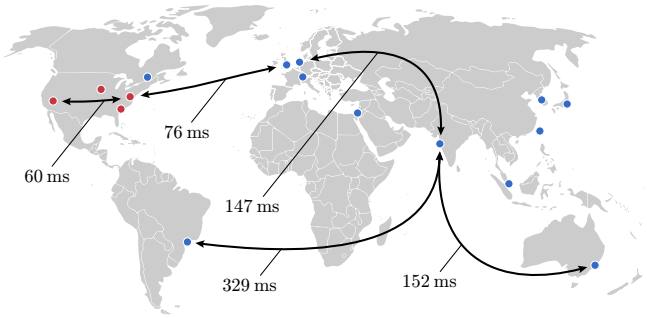

Figure 1: Datacenter Locations and Latency.

Comparison to DKLS18/19. To provide a point of comparison and illustrate the impact of network latency, we also benchmarked the 2-of-n [3] and t-of-n [4] threshold ECDSA signing protocols of Doerner et al., using their own proof-of-concept implementations.[7] Since this implementation uses the same underlying cryptographic libraries and multithreading techniques as our own, and it was compiled with the same compiler and run in the same environment, we believe this comparison to be fair.We report the results of these benchmarks alongside the timings for our own protocol in Table 1. Since their setup protocol is similar to and interchangeable with ours, we do not measure it.

In both WAN settings, the advantage of our three-round protocol is clear: in the US WAN setting it is a factor of roughly three faster than DKLS19 up to $t = 32$, at which point the advantage tapers due to the influence of other bottlenecks, and in the Global WAN setting, our protocol is four to six times faster across almost the entire range of t, with a slight taper evident only at $t = 256$. In the LAN setting, both our protocol and DKLS18/19 are compute-bound, but theirs requires only a constant number of elliptic curve scalar operations, whereas ours requires $O(t)$: for this reason, their protocol is slightly faster than ours beginning at $t = 16$.

Comparison to CGGMP. As discussed in Section 1.1, many recent threshold ECDSA protocols use Paillier-based multiplication strategies that require expensive zero-knowledge proofs to avoid concrete attacks. In Table 3, we present *compute-only* wall-clock times from an implementation of the CGMPP [13] protocol by the Taurus group.[8] These benchmarks were performed with all parties running in parallel on a single machine, containing a 16-core AMD Ryzen 9 7950X clocked at 4.5 GHz and 64GB of memory, and running Linux kernel 5.15.0-83-generic. In the two-party LAN context, the CGGMP protocol costs 600 times more than ours, and our advantage only increases with the number of parties. Since their protocol also requires more rounds than ours, we expect the gap to widen as latency increases as well. In fact, even with 256 parties spread across five continents, our protocol is twice as fast as CGGMP is with two parties on neighboring cores of the same CPU.

7. Available: https://gitlab.com/neucrypt/mpecdsa
8. Available: https://github.com/taurusgroup/multi-party-sig/

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

[1] J. Doerner, Y. Kondi, E. Lee, and abhi shelat, "Threshold ecdsa in three rounds," Cryptology ePrint Archive, Paper 2023/765, 2023, https://eprint.iacr.org/2023/765.

[2] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits, "Low-bandwidth threshold ECDSA via pseudorandom correlation generators," in *43rd S&P*, 2022.

[3] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *39th S&P*, 2018.

[4] ——, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *40th S&P*, 2019.

[5] 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.

[6] L. T. A. N. Brandão and R. Peralta, "NISTIR 8214c ipd NIST first call for multi-party threshold schemes (initial public draft)," in *Computer Security Resource Center*, 2023.

[7] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, no. 1, 2021.

[8] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT*, 2001.

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

[10] J. Bar-Ilan and D. Beaver, "Non-cryptographic fault-tolerant computing in constant number of rounds of interaction," in *8th PODC*, 1989.

[11] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *25th CCS*, 2018.

[12] J. Groth and V. Shoup, "Design and analysis of a distributed ECDSA signing service," Cryptology ePrint Archive, Paper 2022/506, 2022. [Online]. Available: https://eprint.iacr.org/2022/506

[13] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *27th CCS*, 2020.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

Protocol	Network	2	3	4	8	16	32	64	128	256
Our Signing	LAN	**4.0**	**6.1**	**8.4**	**19.0**	39.6	81.3	169.5	334.2	686.2
DKLS18/19 Signing	LAN	5.1	9.8	11.6	22.4	**36.0**	**65.0**	**146.1**	**268.1**	**572.7**
Our Signing	US WAN			**105.3**	**108.2**	**113.7**	**125.8**	**183.0**	**348.6**	**749.8**
DKLS19 Signing	US WAN			270.0	289.1	298.8	326.3	385.7	538.1	1044.2
Our Signing	Global WAN					**494.9**	**503.6**	**532.0**	**623.1**	**1,326.4**
DKLS19 Signing	Global WAN					2203.7	2716.5	3077.3	3335.2	3,777.3
Our Setup	LAN	70.7	139.2	258.5	559.9	1,098.3	2,245.9	4,497.6	8,972.3	18,754.0
Our Setup	US WAN			305.7	526.0	1,071.7	2,178.5	4,426.4	9,080.2	18,428.6
Our Setup	Global WAN					1,350.9	2,455.4	4,723.5	9,510.3	18,802.8

TABLE 1: Wall-Clock Times in Milliseconds, for Our Protocols and the DKLS18/19 Signing Protocols with a Varying Number of Participants. Reported timings are averages over all parties and samples. We benchmark only our own setup protocol, since it is interchangeable with the DKLS19 Setup Protocol.

Protocol	2	3	4	8	16	32	64	128	256
Our Signing	53	106	159	371	796	1,646	3,346	6,746	13,547
Our Setup	41	83	125	295	646	1,380	2,972	6,662	16,047

TABLE 2: KiloBytes Transmitted per Party, for Our Protocols with a Varying Number of Participants. These values include protocol overheads (e.g., session IDs), but do not include authenticated channel overheads.

Protocol	2	4	8
CGGMP Signing	2,488	3,561	7,214
CGGMP Setup	3,071	3,999	7,704

TABLE 3: Compute-Only Wall-Clock Times in Milliseconds, for the CGGMP Protocol with a Varying Number of Participants.

[14] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold EC-DSA," in *23rd PKC*, 2020.

[15] I. Haitner, Y. Lindell, A. Nof, and S. 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. [Online]. Available: https://eprint.iacr.org/archive/2018/987/20230529:135032

[16] Y. Lindell, "Simple three-round multiparty schnorr signing with full simulatability," Cryptology ePrint Archive, Paper 2022/374, 2022. [Online]. Available: https://eprint.iacr.org/2022/374

[17] J. Nick, T. Ruffing, and Y. Seurin, "Musig2: Simple two-round schnorr multi-signatures," in *CRYPTO*, 2021.

[18] M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu, "Better than advertised security for non-interactive threshold signatures," in *CRYPTO*, 2022.

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

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

[21] M. Cerecedo, T. Matsumoto, and H. 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.

[22] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergaard, "Fast threshold ECDSA with honest majority," 2020.

[23] P. D. MacKenzie and M. K. Reiter, "Two-party generation of DSA signatures," in *CRYPTO*, 2001.

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

[25] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," 2016.

[26] D. Boneh, R. Gennaro, and S. Goldfeder, "Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security," 2017.

[27] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft, "Efficient RSA key generation and threshold paillier in the two-party setting," in *CT-RSA*, 2012.

[28] M. Chen, R. Cohen, J. Doerner, Y. Kondi, E. Lee, S. Rosefield, and a. shelat, "Multiparty generation of an RSA modulus," in *CRYPTO*, 2020.

[29] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang, "Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority," 2021.

[30] Y. Lindell, "Fast secure two-party ECDSA signing," in *CRYPTO*, 2017.

[31] N. Gilboa, "Two party RSA key generation," in *CRYPTO*, 1999.

[32] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, 2003.

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

[34] L. Roy, "SoftSpokenOT: Communication-computation tradeoffs in OT extension," in *CRYPTO*, 2022.

[35] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *25th CCS*, 2018.

[36] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *CRYPTO*, 2005.

[37] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *19th STOC*, 1987.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

[38] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security," *Theoretical Computer Science*, vol. 939, 2023.

[39] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from DDH," in *CT-RSA*, 2015.

[40] H. W. H. Wong, J. P. K. Ma, H. H. F. Yin, and S. S. M. Chow, "Real threshold ECDSA," in *NDSS*, 2023.

[41] N. P. Smart and Y. Talibi Alaoui, "Distributing any elliptic curve based protocol," in *IMA International Conference on Cryptography and Coding*, 2019.

[42] A. P. K. Dalskov, C. Orlandi, M. Keller, K. Shrishak, and H. Shulman, "Securing DNSSEC keys via threshold ECDSA from generic MPC," in *25th ESORICS*, 2020.

[43] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.

[44] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *EUROCRYPT*, 2011.

[45] D. Masny and P. Rindal, "Endemic oblivious transfer," in *26th CCS*, 2019.

[46] R. Canetti, P. Sarkar, and X. Wang, "Blazing fast OT for three-round UC OT extension," in *23rd PKC*, 2020.

[47] Y. Kondi and a. shelat, "Improved straight-line extraction in the random oracle model with applications to signature aggregation," in *ASIACRYPT*, 2022.

[48] I. Haitner, N. Makriyannis, S. Ranellucci, and E. Tsfadia, "Highly efficient OT-based multiplication protocols," in *CRYPTO*, 2022.

[49] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd FOCS*, 2001.

[50] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *34th STOC*, 2002.

[51] J. Doerner, Y. Kondi, E. Lee, a. shelat, and L. Tyner, "Threshold BBS+ signatures for distributed anonymous credential issuance," in *44th S&P*, 2023.

Appendix A.
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 A.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*

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

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

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 A.2. $\mathcal{S}^{\mathcal{A}}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$

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

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

3) On receving $(\texttt{abort}, \texttt{sid})$ from $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, send $(\texttt{abort}, \texttt{sid})$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.
4) On receiving $(\texttt{adv-poly}, \texttt{sid}, \{\check{p}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\check{P}(j)\}_{j\in\mathbf{P}^*})$ 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
   - $(\texttt{public-key}, \texttt{sid}, \mathsf{pk})$ directly from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$
   - $(\texttt{adv-poly}, \texttt{sid}, \{\check{p}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\check{P}(j)\}_{j\in\mathbf{P}^*})$ 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 $(\texttt{hon-poly}, \texttt{sid}, \{\hat{P}(i)\}_{i\in[n]\setminus\mathbf{P}^*}, \{\hat{p}(j)\}_{j\in\mathbf{P}^*})$ directly to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{RelaxedKeyGen}}(\mathcal{G}, n, t)$, and store $(\texttt{public-key}, \texttt{sid}, \mathsf{pk}, \{P(i)\}_{i\in[n]})$ in memory.
6) On receiving $(\texttt{release}, \texttt{sid}, i)$ or $(\texttt{abort}, \texttt{sid}, i)$ 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 $(\texttt{public-key}, \texttt{sid}, \mathsf{pk})$ from $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_j$ for $j \in \mathbf{P}^*$, store $(\texttt{sk-released}, \texttt{sid}, j)$ in memory.
8) Initialize the blacklist for sid to be empty.

Signing:

9) On receiving $(\texttt{sig-req}, \texttt{sid}, \mathsf{sigid}, j, m_j)$ 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 $(\texttt{public-key}, \texttt{sid}, \mathsf{pk}, \{P(i)\}_{i\in[n]})$ and $(\texttt{sk-released}, \texttt{sid}, i)$ 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 $(\texttt{sign}, \texttt{sid}, \mathsf{sigid}, m_h)$ to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ on behalf of $\mathcal{P}_i$ and send $(\texttt{proceed}, \texttt{sid}, \mathsf{sigid}, i)$ directly to $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$. If $(\texttt{signature}, \texttt{sid}, \mathsf{sigid}, (s, r^{\mathsf{x}}))$ 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 $(\texttt{failure}, \texttt{sid}, \mathsf{sigid})$ 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 $\mathcal{F}_{\mathsf{Com}}$
   - $(\texttt{check-adjust}, \texttt{sid}, \mathsf{sigid}, \boldsymbol{\Gamma}^{\mathsf{u}}_{j,}, \boldsymbol{\Gamma}^{\mathsf{v}}_{j,i}, \boldsymbol{\psi}_{j,i}, \mathsf{pk}_j)$ on behalf of $\mathcal{P}_j$

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

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

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

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

$$\begin{aligned}
\phi_i &:= \boldsymbol{\psi}_{i,h} + \boldsymbol{\chi}_{i,h} \qquad \text{for } i \in \mathbf{C} \\
\hat{u} &:= \sum_{i \in \mathbf{C}} \begin{pmatrix} \mathbf{a}^{\mathsf{u}}_{i,h} \cdot \left(\sum_{k \in \mathbf{P}^{-h}} \phi_k + \boldsymbol{\psi}_{h,i} \right) \\ + \mathbf{c}^{\mathsf{u}}_{i,h} + \mathbf{d}^{\mathsf{u}}_{i,h} \end{pmatrix} \\
&\quad + \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta^{\mathsf{u}}_j + r_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right) \\
\hat{v} &:= \sum_{i \in \mathbf{C}} \begin{pmatrix} \mathbf{a}^{\mathsf{v}}_{i,h} \cdot \left(\sum_{k \in \mathbf{P}^{-h}} \phi_k + \boldsymbol{\psi}_{h,i} \right) \\ + \mathbf{c}^{\mathsf{v}}_{i,h} + \mathbf{d}^{\mathsf{v}}_{i,h} \end{pmatrix} \\
&\quad + \sum_{j \in \mathbf{H} \setminus \{h\}} \left(\delta^{\mathsf{v}}_j + \mathsf{sk}_j \cdot \sum_{i \in \mathbf{C}} \phi_i \right) \\
\hat{w} &:= \mathsf{SHA2}(m_h) \cdot \sum_{k \in \mathbf{P}^{-h}} \phi_k + r^{\mathsf{x}} \cdot \hat{v} \\
w_h &:= s \cdot u_h + s \cdot \hat{u} - \hat{w}
\end{aligned}$$

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 $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

Our sequence of hybrid experiments begins with the real world

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

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

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

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.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

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 $j \in \mathbf{H}\backslash\{h\}$. If there exists some $j \in \mathbf{H}\backslash\{h\}$ 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.[9]

Summing and rewriting, we have

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

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, $\mathcal{S}$ 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

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

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

9. 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.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

it computes if they misbehave, conditioned on the fact that the consistency check passes.

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

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

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

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

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

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

Note that because $\mathbf{c}^{\mathsf{u}}$, $\mathbf{d}^{\mathsf{u}}$, $\mathbf{c}^{\mathsf{v}}$, and $\mathbf{d}^{\mathsf{v}}$ are uniformly sampled subject to constraints upon their component-wise sums, δ_j^{u} and δ_j^{v} are uniform when considered indepently of the view of $\mathcal{P}_h$. Note also that when the consistency check passes for $\mathcal{P}_h$, we can be sure that $\mathbf{a}_{i,h}^{\mathsf{v}} \cdot G = \mathbf{pk}_{i,h}$ 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 $\delta_j^{\mathsf{u}} \leftarrow \mathbb{Z}_q$ and $\delta_j^{\mathsf{v}} \leftarrow \mathbb{Z}_q$ for $j \in \mathbf{H} \setminus \{h\}$ uniformly, calculates $\hat{u}$ and $\hat{w}$ as defined above, and then computes

$$w_h \coloneqq s \cdot u_h + \sum_{k \in \mathbf{P}^{-h}} \left(s \cdot \hat{u}_k - \hat{w}_k \right)$$

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 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

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

and by transitivity we also have

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

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 A.1 holds. □

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.

Appendix B.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

The paper presents a three round protocol for threshold ECDSA signing that achieves UC security with abort against a malicious dishonest majority, assuming ideal commitments and two-party multiplication.

B.2. Scientific Contributions

Provides a Valuable Step Forward in an Established Field

B.3. Reasons for Acceptance

This paper provides a valuable step forward in an established field: ECDSA is one of the most popular signature schemes in practical use, and threshold signing protocols for ECDSA have numerous applications. The proposed protocol achieves state-of-the-art efficiency for the dishonest majority setting. This is achieved via a new, simple design based on so-called ECDSA tuples.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on March 17,2025 at 14:26:45 UTC from IEEE Xplore. Restrictions apply.