Evaluation of threshold ECDSA protocols applied to cryptocurrency

Mary McCready* Rohith Krishna Papani † Parth Kachhadia‡

December 4, 2022

Abstract

Elliptic Curve Digital Signature Algorithms (ECDSA) are popular amongst blockchain networks, such as Bitcoin and Ethereum, because of their security and efficiency. The distributed variant of this primitive, Threshold ECDSA, improves security by sharing the common key among multiple parties and requiring a threshold of parties to decrypt or sign a message. Different modern cryptographic constructions were assessed for security, computation and communication costs incurred due to reliance on heavy zero-knowledge proofs, and practicality. These protocols show trade-offs between cost, speed and the functionalities they provide, such as support for a maximum number of parties, ability to identify a corrupted party, verification error aborts, and interactive versus non-interactive signing. All of these constructions rely on different standard and non-standard assumptions to support the theoretical security proofs they provide. Some protocols can actually be applied in real-time applications in very specific scenarios.

1 Introduction

Decentralization and security are primary benefits that make cryptocurrencies an attractive alternative to standard currency. Many cryptocurrencies are built on blockchain technology, which serves as a decentralized ledger, and are secured with public key cryptography. Popular cryptocurrencies, such as Bitcoin and Ethereum, use Elliptic Curve Digital Signature Algorithm (ECDSA) to create keys and sign transactions. The secp256k1 curve is commonly used in the blockchain space and has a security level of 256 bits.

Put very simply, transactions have a sender and receiver, each with a key pair. The sender encrypts the transaction details with the receiver's public key and applies their own digital signature before sending to the receiver. The receiver verifies the sender's digital signature and then decrypts the transaction with their own private key. This method ensures that the sender is the true owner of the funds, and the receiver is the intended recipient, without disclosing additional information about either. Just as impersonation and fraud can facilitate theft of standard currency, cryptocurrencies can be vulnerable to transaction malleability where a sender's signature is forged or corrupted before it is verified. To mitigate such attacks, Gennaro et al.[GGN16] proposed the use of *threshold* ECDSA, which would eliminate a single point of failure by splitting a key amongst several participants. A threshold of participants are needed to produce a signature, increasing the attack surface, yet producing a signature that is identical to those produced from a centralized or non-threshold signature scheme.

2 Problem Setup and Threat Model

Although there are clear security benefits, early threshold ECDSA protocols were too computationally expensive to be practical for cryptocurrency. Each signing operation requires zero-knowledge proofs or large-modulus exponentiation[Lin17]. These performance limitations make threshold ECDSA an unrealistic option for cryptocurrencies; however, they have sparked a renewed interest in threshold ECDSA research. In this paper, we review seven new threshold ECDSA protocols and an aggregate signature alternative that seek to improve upon the efficiency and security of early protocols. We derived a general protocol and set of definitions, outlined in this section, from the work of Gennaro et al. [GGN16] to serve as the baseline by which we compare the new protocols.

As in Gennaro et al.[GGN16], we consider an attack landscape where there are n players connected by a network of point-to-point channels and a broadcast channel. Players participate in two phases, key generation and signature generation. In threshold ECDSA, there is a threshold t of n players, and $t + 1$ players are necessary to provide a valid signature.

*MS Computer Science, 2nd year

†MS Computer Science ,1st Year

‡Here, write what year of your graduate studies you are and in which program you are enrolled in, e.g. CSE Master's 2nd year

Key generation

During key generation, players jointly generate an ECDSA key pair with a single public key and a private key shared amongst players. The participants on the network contribute toward the randomness of the private key, let's say K. Player P_i has share k_i, along with some other values that are essential for partial signature generation. The players then verify their shares using a verifiable signature scheme to check for the consistency of their shares. A secret sharing scheme could be applied to ensure that the secret key is hidden and shares that can be used to reconstruct the secret key can be shared among the participants. For example:

1. Player i chooses random numbers k_i from Z_q and it lies on the elliptic curve chosen
2. Now the player computes $y_i = g^{k_i}$ where g is the generator of the elliptic curve.
3. Then, a non-standard primitive (commitment scheme) is used to ensure player i does not repudiate the k_i chosen in the initial stage of key generation.
4. Players then share outputs from commitment scheme, their share of public key y_i, encryption of their share k_i.

Each player now has the capability to compute the public key $y = \prod_{i=1}^{n} y_i$, and the consistency of everyone's share could be checked using Feldman VSS.

Signature generation

Each player now has a share of the secret key along with some essential values. The players who have verified their shares perform advanced mathematical concepts, such as multiplicative-to-additive share conversion via oblivious transfers or calculating Lagrangian coefficients, and non-standard primitives, such as commitment scheme, to obtain their share of the signature. For example, player i has the share s_i where $\sum_{i=1}^{t+1} s_i = S$. Once $t+1$ players put together their share of the signature, and the signature is verified, the transaction will be signed.

In ECDSA, the signature is computed $S = k^{-1}(H(m) + r \cdot x) \bmod q$, where k is the nonce or instance key selected from Z_q, m is the message, $H()$ is a one way hash function, $H(m)$ is the hash of the message, x is the secret key, r is generated by using k and generator point, q is the order of the elliptic curve and generator point of the elliptic curve.

In threshold cryptography every user holds a partial signature of form $s_i = k_i^{-1}(H(m) + r_i \cdot x_i \bmod q$. The signature share is tested for validity before revealing it other parties to compute the complete signature.

The above shown simple summation of signatures ($\sum_{i=1}^{t+1} s_i = S$) is only provided to demonstrate how threshold cryptography works. Throughout our literature review, we noticed several methods used to aggregate all the partial signatures in a manner that is tangible to implement in a real world scenario and less computationally intensive. These algorithms usually require a large number of communications for each transaction participant over several rounds while also performing computations for key and signature generation. This leads to high communication costs when implemented in the real world when the threshold number of parties of a transaction verification is high and there are more of such transactions.

Security definitions

A summary of the formal definition for secure signature scheme is that a signature is unforgeable if no adversary who is given a public key and the signatures of other messages can produce the signature on a new message. For threshold secret sharing, (t, n)-threshold secret sharing of t or less values must reveal no information about the secret. A threshold signature scheme is unforgeable if no malicious adversary who corrupts at most t players can produce the signature of any new message having viewed the key generation and signature protocols on input messages which the adversary chose. As we examine other protocols, we noticed different types of game-based or simulation-based definitions, assumptions based on security proofs of primitives, and information theory-based definitions, including adversaries with unlimited compute power and time, against non-standard primitives.

Threat model

As threshold ECDSA signature relies on t-parties to successfully generate a signature, an attacker can interfere during the execution in different ways, which can cause failure, abort of execution or can provide the attacker an advantage over the signature scheme. As mentioned in previous sections, threshold schemes rely heavily on communication between parties to share information required to compute a threshold signature. An attacker can perform man-in-the-middle attacks to obtain such information. One advantage of threshold schemes is that an attacker must corrupt the majority ($> t$) of n participants to generate valid signatures. In some cases, an attacker can simply disrupt the threshold signature scheme, if it serves her objective by proactively sharing false information while execution, as some threshold protocols rely on tight verification and aborts if not verified. In two-party schemes, an attacker can use deceptive methods to obtain secret information from another participant and gain control of the threshold signature.

Protocols reviewed in this paper apply various techniques to overcome some of the limitations faced by threshold ECDSA signature schemes.

3 Literature Review/Theory/Construction/Analysis

The protocols we reviewed were published in top venues from 2017 to 2022. We found that the later publications were often derived from or compared to the earlier publications (or similar works from the same authors).

Two-party threshold protocols

The oldest work we reviewed was of a two-party threshold ECDSA (meaning there are only two players P_1 and P_2) developed by Yehuda Lindell in 2017[Lin17]. We mentioned previously that threshold ECDSA is impractical due to the computational expense of zero-knowledge proofs. Lindell sought to improve performance by having a more complex zero-knowledge proof in key generation protocol, which is only run once, instead of the signing protocol which, in this case, has four rounds. During key creation, P_1 computes two Elliptic curve multiplications, a Paillier public-key generation, and a Paillier encryption. P_1 also provides more expensive expensive Paillier public-key and zero-knowledge proof for the PDL language proofs. P_2 only computes two Elliptic curve multiplications. Both parties are once a prover and once a verifier for two Schnorr zero-knowledge proofs of knowledge for discrete log. In the signature generation protocol, P_1 computes seven Elliptic curve multiplications and one Paillier decryption, while P_2 computes five Elliptic curve multiplications, a Paillier encryption, a Paillier homomorphic scalar multiplication, and a Paillier homomorphic addition. The PDL proof realizes a connection between properties of Paillier encryption and discrete log that enable a valid signature as long as there is proof that the Paillier key was generated correctly and the encryption is of a valid share of the key. Thus, corruption is either detected by the zero-knowledge proof or through signature validation.

Lindell[Lin17] used experiments to provide game-based and simulation-based definitions that prove the security of their protocols. The experiment includes concurrent signing, and all executions will abort if corruption is detected. They could prove security all but in the case where P_2 is corrupted and simulator must "guess" when to abort. Instead of adding more zero-knowledge proofs, Lindell identified a new, non-standard assumption based on giving the adversary a specific oracle. In 2019, Castagnos et al.[Cas+19] sought to improve on Lindell's[Lin17] work by developing a two-party protocol that did not require interactive assumptions. They found that they could replace Paillier encryption with hash proof systems and enjoy the same homomorphic benefits while eliminating the need for additional zero-knowledge proofs. This is achieved through two new assumptions: decomposablity and double encoding. Decomposablity is a property of projective hash functions that ensures separation between the predictable and random parts of a hash. Double encoding handles the fact that a successful completion of the protocol, or an abort as a result of corruption, is useful information to an adversary by reducing the probability of multiple invalid encryptions. The security of these protocols was proven via indistinguishable game-definition. While proven more secure and efficient than Lindell's protocol, the computational expense of key generation is not ideal.

In 2021, Deng et al.[Den+21] acknowledged benefits and limitations of both [Lin17] and [Cas+19] protocols, and further introduce a promise Σ-protocol which improves efficiency issues in [Cas+19]'s two-party protocol. The promise Σ-protocol is used to prove equality of messages and homomorphic operations when a combination of ElGamal and CL encryption is used. [Den+21] modified other popular two-party and multiparty threshold ECDSA protocols, replacing heavy zero-knowledge proofs with the promise Σ-protocol. More specifically, during key generation, parties encrypt using ElGamal, which allows for proof via promise Σ-protocol during signing. [Den+21] then compared the performance of their protocols against [Lin17] and [Cas+19], which showed that the promise Σ-protocol was both faster and cheaper than the other two-party protocols.

Multiparty threshold protocols

Now we examine a multiparty threshold ECDSA introduced by Gennaro and Goldfeder[GG18], which also seeks to minimize the need for zero-knowledge proofs and often compares to the early work of Lindell[Lin17]. [GG18] propose splitting the keys among n players on the network and design an algorithm such that at any point a minimum of $t+1$ players are required for signing the transaction. [GG18] address an issue raised in a previous iteration of the protocol that required $2t+1$ players to complete the signature (a and b were free terms of polynomial of order t, so when secret key $k = a \cdot b$, $2t+1$ players are required to sign the transaction), which increases the attack surface. To circumvent this problem, [GG18] use a multiplicative-to-additive share conversion method, where we find the shares of each set of players in such a way that only $t+1$ players are required to sign the transactions. Let a and b be the secret shares of two parties such that secret key $s = a \cdot b$, we can now calculate a', b' such that $a' + b' = s$ using the properties of Paillier encryption, which includes homomorphic addition and multiplication. While converting from multiplicative-to-additive shares, Paillier encryption with modulus N is used to calculate the shares, which can cause some confusion between the modulus of the Paillier encryption and the modulus over which shares operate, which is q because the number is chosen at random from Z_q. [GG18] use a very large value of N to ensure operations modulo N do not wrap around and are consistent over integers. Feldman VSS is used during the key generation phase to ensure players choose secret keys in Z_q. In a verifiable secret sharing scheme, auxiliary information allows players to verify that their shares are consistent which prevents the need for costly zero-knowledge proofs.

During signature generation, a multiparty computation technique referred to as *"Beaver's Trick"* is used for preprocessing inefficient multiplication protocols. A Beaver triple $(k, \sigma, k \cdot \sigma)$, denoted as δ, is made public. A shared signature is computed using Beaver's trick and multiplicative-to-additive share conversion. To prevent leakage of information about shares of honest players when there exists a dishonest player, the scheme includes a distributed randomized signature verification. If the test passes, the shares can be released, else the protocol is aborted. This paper is limited to static corruptions where an adversary must choose which players to corrupt at the beginning of the protocol, which might not be the case in a real-world scenario.

Canetti et al.[Can+20] further builds on the work of [GG18] in 2020, by developing two new protocols for any number of signatories and any threshold. The protocols featured non-interactive signing, round-minimal interactive signing, proactive key refresh, and identifiable aborts, along with improved efficiency and security guarantees not seen in previous protocols. The protocols we have reviewed thus far generally have two phases: key generation and signature generation. The protocol introduced by [Can+20] consists of four phases: key generation, refresh of secret key shares, signature preprocessing, and signature computation. During key generation, each player samples a local random secret share of the master secret key and then reveals their computation and a Schnorr non-interactive zero-knowledge proof of knowledge. In the key refresh phase (which includes computation of auxiliary information), each player generates a Paillier key and broadcasts that with a non-interactive zero-knowledge proof. Each player then performs a computation and a discrete log proof of knowledge. The players update their key shares if all proofs are valid. In the signature preprocessing phase, each player commits to their shares needed of the pre-signing data. This eliminates the need for zero-knowledge proofs that are needed when Paillier encryption is used for ECDSA. In the signing phase, a signature is generated from the pre-signing data.

Doerner et al.[Doe+19] builds on his previous work, a 2-of-n threshold ECDSA under naive assumptions of ECDSA curve, which we did not review but is similar to that of [GG18]. The algorithm presented in Doerner's previous work is limited to the two-party threshold and uses Diffie Hellman key exchange for signature generation. [Doe+19] extends the previous work by providing support for an arbitrary threshold

and proving its security against dishonest players that count to one less than the threshold. The key generation protocol is costly in a t-of-n case ($t-1$ rounds between players) if Diffie Hellman key exchange is applied between every two parties. Instead, [Doe+19] utilize multiplicative-to-additive shares and reduce the number of rounds to $log(t)+2$ or 1 depending on the case. This protocol replaces the key exchange component with a multiparty multiplication.

When t participants agree to sign a transaction, they calculate the additive share using Lagrange coefficients. Each participant chooses a multiplicative share and contributes it towards generation of the instance key k. The protocol utilizes the concept of multiplicative-to-additive share conversion using a t-party multiplication protocol, and the participants get additive shares of k and a multiplicatively padded additive share of k^{-1}. Now a GMW-style multiplication of the additive share of secret key and the share of instance key is followed to generate the additive share of an intermediate value that is used later during signature generation of each participant. A consistency check ensures the inputs are consistent in both the above defined multiplication processes. Once each participant passes the consistency check, they compute their partial share of the signature which is summed with all t participants to complete the signature, which can be verified by standard verification algorithms. This protocol allows for multiple parties to process in parallel, rather than in sequence, further improving efficiency over previous works of Doerner et al. when extended to the current scenario. This protocol does not use Paillier, but uses multiplication-by-oblivious-transfer instead. Oblivious transfer extension requires no new assumptions as opposed to Paillier encryption which introduces Decisional Composite Residuosity and the multiplication carried out costs only a few milliseconds to execute. The cryptographic assumptions, such as hardness of Diffie Hellman, are also native to the curve.

Most recently, in 2022, Abram et al.[Abr+22] introduced silent preprocessing to further improve efficiency. The protocol relies on generating correlated randomness in significant quantity. Each participant generates multiplicative triplets, which starts the execution. To generate such tuples, [Abr+22] defined a new system called pseudo-random correlation generator which relies on the distributed point functions (DPF) to provide successful results. This unique scheme allows reduction in the number of rounds in preprocessing phase. This protocol relies on a module-LPN (Learning Parity with Noise) assumption with static leakage, which is a variant of the LPN assumption over polynomial ring. Execution of this protocol can be varied, as a one-round presigning phase with non-interactive signing or non-interactive presigning phase with two-rounds of signing. Since multiplicative triplets are generated via PCG, verification can be done for only a single value. All of these combined allow for an overall efficient threshold ECDSA signature scheme.

Aggregate Signature protocol

Lastly, we reviewed the work of Zhao[Zha19], who offers an effective Aggregate Signature scheme as an alternative to threshold ECDSA. As mentioned earlier in the review, there are some limiting elements like validating all the updates to ledger, size of signatures and computation costs for verification which lead to scaling issues in cryptocurrency. The aggregate signature in general, when implemented in a blockchain network, can provide faster verification and improved scaling ability through reduced need for storage and better throughput. It has the potential to make the existing blockchain systems more efficient, as communication and storage costs tend to be heavier than computation costs.

The aggregate signature scheme proposed by [Zha19] can mitigate these deficiencies or bottlenecks faced that make threshold ECDSA impractical for cryptocurrency. Individual signatures on a message can be collected non-interactively and condense to a compact aggregate signature. [Zha19] uses a T-signature scheme based on discrete logarithm problem proposed in their earlier works, which relies on cryptographic hash functions to sign and verify signature. As an extension to this, **Agg.** and **AggVerify** algorithms were proposed. **Agg.** enables condensing individual sign-on message under public-key into a compact aggregate signature. The aggregate T-signature scheme relies on the non-malleable discrete logarithm assumption and security proof is provided in the random oracle model. This protocol adopts the Merkle Patricia Tree based implementation for aggregating signatures. ECDSA signing involves a non-linear combination of ephemeral and static secret keys and sometimes involves performing mathematical operations that introduce new assumptions not native to the elliptic curve, which is mitigated with aggregate signatures.

4 New Construction

In this section, we outline four new constructs that will improve the protocols we reviewed:

1. In the implementation details of [Doe+19], we observed that the session and party IDs are transmitted along with the message to the functionalities. By adding a new functionality to this protocol we can find the dishonest party which is a major extension to existing protocols.

2. While performing benchmarks for [Can+20], we observed that they adopt a commitment scheme using the SHA3 hashing algorithm, which is susceptible to collision attacks like practical collision and near collision attacks. We suggest using SHA256 to improve efficiency and security over SHA3.

3. We propose using an alternative commitment scheme called Pedersen's commitment scheme in [Can+20]. Pedersen's commitment scheme focuses on perfect hiding as opposed to hash-based commitments, which emphasize on computational binding. The shares we are concerned with protecting in a transaction are refreshed for the user when signing is over, and Pedersen's commitment scheme would ensures that no information is leaked during the signing process, which is an improvement over hash-based commitments.

4. One novel construction would be to take advantage of more efficient elements from different constructions discussed in the review. Pseudorandom correlation generators (PCGs) can be used to generate n secret correlated seeds, one for each participant. These seeds then can be expanded to secret share as per [Abr+22]. The shares can be used to generate a signature. In the next step, [Zha19]'s **Agg.** algorithm can be used to generate a final aggregate signature with n parties. For verification of each party's sign, only x's share of the $ECDSA$-tuple can be used, as in [Abr+22]. **AggVerify** can be modified in a similar way to give full construction of an efficient cryptographic *ECDSA Aggregate Signature Scheme*.

5 Evaluation

Each publication we reviewed presented benchmark results or performance comparisons except for Canetti et al.[Can+20], making their protocol a good candidate for our own benchmark evaluation. In this section, we give an overview of the implementation of [Can+20] developed by a digital asset platform company, Taurus[Ham21]. The [Ham21] implementation of [Can+20] consists of several functions based on the phases from the paper:

- Generation of a new ECDSA private key which is shared amongst all participants
- Online signature generation (4 rounds)
- Refreshing of existing ECDSA private key shares
- Signature preprocessing
- Non-interactive combining of presignatures (7 rounds)

Table 1 shows the times for each phase in milliseconds for a multiparty scenario, which were run on an Apple M2 8 Core 3500 MHz machine. We kept the default message to be signed, *hello*, and default security parameters. We varied the values for the total number of players, n, and the threshold of signers, t. It is clear that signing time increases as participants and thresholds increase. At $n = 50$, our machine did not appear to have enough capacity. We attempted smaller combinations of (n, t) between 10 and 50 without success. This further demonstrates the computational cost of threshold ECDSA.

Each signature generation relies on a different verification process. The protocols provide a tradeoff between the number of rounds to generate a signature and the computational and communication overhead for the identification of corrupted signatories. It is clear that the non-interactive presigning protocol is more efficient when there is a need to sign multiple messages. Since presigned shares are stored and reused, the presigning phase does not run for each new message, saving time over interactive signing methods. We observed that parallel processing was implemented to speed up the heavy processes. We also observed that in the implementation of this protocol, the timing leaks were prevented by choosing a constant time arithmetic.

Table 1: Benchmark of [Can+20] via [Ham21]					
Participants, Threshold (n,t)	Key Gen	Signing	Refresh	Presig	Combine presig
(5,3)	1102 ms	407 ms	846 ms	3606 ms	2.49 ms
(10,7)	2225 ms	1510 ms	2565 ms	1510 ms	9.35 ms
(50,40)	244067 ms	?	?	?	?

6 Conclusions

Although there is a direct application of threshold ECDSA to cryptocurrency transactions, cryptocurrency exchanges and platforms have been slow to adopt threshold ECDSA due to their computational expense. Efficiency issues are largely due to the need for complex zero-knowledge proofs, and current research has applied several methods in an attempt to reduce the number of zero-knowledge proofs or eliminate the need for them entirely. Techniques to reduce the need for zero-knowledge proofs through commitment schemes, including proof the connection between Paillier encryption and discrete log[Lin17] or replacing Pallier encryption with other homomorphic encryptions such as hash proof systems [Cas+19] or promise Σ-protocol [Den+21]. Authors replaced inefficient mathematical operations with advanced mathematical concepts like oblivious transfers[Doe+19], multiparty computation technique like *"Beaver's Trick"*[GG18], and so on. [Doe+19] evaluated performance over WAN (apart from LAN) with nodes setup across the world. The results indicated improved performance over other protocols, which suggests that we might see a more extensive adoption of threshold cryptography in the blockchain space.

Protocols that leverage some type of preprocessing, such as [Can+20]'s signature preprocessing phase, can further eliminate the need for zero-knowledge proofs. [Abr+22] proposes a distributed point function based on the concept of learning parity with noise, which reduces the number of rounds in the preprocessing stage. However, preprocessing may introduce storage concerns and attacks against static corruptions, which is where protocols like [GG18],[Doe+19] had added value. Alternative techniques to threshold ECDSA, such as the protocol presented by [Zha19] based on the concept of aggregate signatures, have the potential to reduce signature and verification times that make threshold ECDSA an impractical option for cryptocurrency.

The current research on threshold ECDSA seems promising, and we expect that the momentum of research on viable threshold ECDSA for cryptocurrency will continue. The protocols are largely theoretical, and we are interested to know how well these constructions can be scaled to support real-time applications. We also have questions around whether the increased security is worth the computation or storage costs, or if there are other methods outside of digital signatures that can be employed to address security concerns.

Resources

[Abr+22] Damiano Abram et al. "Low-bandwidth threshold ECDSA via pseudorandom correlation generators". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 2554–2572.

[Can+20] Ran Canetti et al. "UC non-interactive, proactive, threshold ECDSA with identifiable aborts". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1769–1787.

[Cas+19] Guilhem Castagnos et al. "Two-party ECDSA from hash proof systems and efficient instantiations". In: *Annual International Cryptology Conference*. Springer. 2019, pp. 191–221.

[Den+21] Yi Deng et al. "Promise $\sum$-Protocol: How to Construct Efficient Threshold ECDSA from Encryptions Based on Class Groups". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2021, pp. 557–586.

[Doe+19] Jack Doerner et al. "Threshold ECDSA from ECDSA assumptions: the multiparty case". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1051–1066.

[GG18] Rosario Gennaro and Steven Goldfeder. "Fast multiparty threshold ECDSA with fast trustless setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1179–1194.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, pp. 156–174.

[Ham21] Hamelink, Adrian and Meier, Lúcás Críostóir and Aumasson, J.-P. *multi-party-sig*. Version v0.3.0-alpha-2021-08-09. 2021. URL: `https://github.com/taurusgroup/multi-party-sig`.

[Lin17] Yehuda Lindell. "Fast secure two-party ECDSA signing". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 613–644.

[Zha19] Yunlei Zhao. "Practical aggregate signature from general elliptic curves, and applications to blockchain". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 2019, pp. 529–538.