Threshold Signatures using Secure Multiparty Computation

G.Tillem O.Burundukov

ING DLT Team
blockchain@ing.com

Abstract

Distributed ledger technology (DLT) provides powerful solutions to achieve decentralized operations, but running a DLT platform securely remains a challenge. One critical problem in such platforms is the secure management of the keys that are used in the DLT operations. Using a cryptographic technique called secure multiparty computation (MPC), it is possible to overcome the key management problem in DLT platforms. An area of MPC, which is called threshold cryptography, facilitates the distribution of a secret key to multiple parties, which eliminates the risks of a single point of failure. In this paper, we explain MPC and threshold cryptography and describe how they can be used in digital signatures. Furthermore, we discuss what additional security mechanisms can be applied to achieve proactive security in threshold signatures. Finally, we share our experience in the implementation of a threshold signature library and discuss the details of implementation.

1 Introduction

Digital signatures play an essential role in the operation of Distributed Ledger Technology platforms by enabling the identification and authentication of the entities involved in the network. A digital signature scheme consists of two keys, where a private key is held by the owner to sign a data, and a public key is used by other parties in the network to validate the signature of the private key holder.

A challenge in the usage of digital signatures is the management of private keys whose leakage to malicious parties could harm the validity of the DLT platform. Therefore, the storage and management of the key should be maintained carefully. Delegating the key management to a trusted party does not overcome the problem. Although the trusted party provides a better protection mechanism, a single point of failure is still an issue.

Recently, researchers proposed a solution to the key management problem in DLT using an old cryptographic technique, secure multiparty computation

(MPC). MPC enables a group of parties who do not necessarily trust each other to perform operations on their private data without revealing any information about their data. Threshold cryptography is an area of MPC which allows the sharing of a private key among multiple parties. In a threshold cryptography scheme, the private key can be revealed only when a specific subset of parties combines their key shares. Usage of a threshold signature scheme thwarts the single point of failure attacks in key management.

Threshold signatures are not the only available solution for the key management problem. An alternative mechanism is the multi-signature approach, where a transaction is signed by multiple parties individually using their signing keys. Similar to threshold signatures, multi-signature prevents the single point of failure attacks. However, the multi-signature approach requires to perform signature verification multiple times since each signer party's signature needs to be validated. On the contrary, using threshold signing, multiple parties can produce a single signature that can be validated efficiently. Another drawback of the multi-signature approach is that it should be performed on-chain while threshold signatures can be operated off-chain.

In this paper, we aim to explain what threshold signatures are and how they can be securely implemented in DLT systems to overcome the key management problem. The outline of the paper is as follows. In Section 2, we explain MPC and threshold cryptography briefly. Then, we describe how a signature scheme can be transformed into a threshold signature scheme based on the proposal of Gennaro and Goldfeder in [13]. In Section 3, we discuss practical security concerns and how proactive security can be achieved in the implementation of a threshold signature protocol. Finally, in Section 4, we describe our implementation of the threshold signature protocol.

2 Secure Multiparty Computation

Secure multiparty computation enables multiple parties to perform a computation on their secret inputs without revealing any party's input to the other ones. The concept is first introduced by Andrew Yao as a secure computation between two parties [24]. In this seminal work, Yao uses the Millionaires' problem as an example to explain how the two parties can decide who is richer without revealing their wealth to each other. Since then, MPC has been investigated in academia from a theoretical perspective regarding its security definitions and feasibility [1, 2, 7, 9, 10, 15, 17]. Advancements in the communication and computation technology and cryptography greatly reduced execution time of MPC protocols making them viable for applications. The last couple of years has seen applications of MPC in auctions [5], data analytics [3, 4], and DLT [12, 11].

A secure protocol that is based on MPC should achieve several properties [16]:

- **Privacy:** The parties should not learn anything from the computation apart from the output of the computation. More specifically, the only

information party A can learn about party B's input is only whatever information that party A can derive from the output.

- **Correctness:** Correctness requirement guarantees that each party receives the correct output as the result of the computation.
- **Fairness:** All parties should be guaranteed to receive the output. If an honest party cannot receive the output of MPC protocol, then the malicious parties in the computation should not also receive it.

In some cases, fairness property can be omitted since guaranteeing in real-life fairness might be costly in terms of performance, or it might be even impossible to achieve it [20].

Secret Sharing An essential primitive to perform MPC among a set of parties is secret sharing. In secret sharing, secret data can be shared among n parties such that any subset of at least $t+1$ parties can reconstruct the secret, but the subsets with t and fewer parties cannot learn anything about the secret. Such a scheme is called (t, n)-threshold scheme. Shamir's scheme [22] is a well-known scheme for secret sharing, which is based on polynomials. A party can generate secret shares of its secret data s using Shamir's secret sharing scheme by generating a random t degree polynomial $p(x) = a_0 + a_1x^1 + a_2x_2 + \cdots + a_tx^t$, where $a_0 = s$ and the coefficient a_is are generated randomly. Then, for n parties, the secret share of each party $i \in [1, n]$ is equal to $P_i = p(i)$. All operations are performed in a prime modulo p, where $p > n, s$.

Threshold Cryptography An notable area of MPC is threshold cryptography, where a set of parties performs certain cryptographic operations while none of them holds the secret key [20]. Instead, the key is distributed among the parties, and it can be used only when a subset of parties size of which is larger than a certain threshold, combines their key shares.

The research on threshold cryptography is applied in different fields such as encryption/decryption operations in symmetric and asymmetric cryptosystems and digital signatures [6]. From a practical side, a prominent application of threshold cryptography is the protection of signing keys that are used in DLT systems. The signing key is not stored by a single party and a transaction on DLT is only approved by a quorum instead of a single party. In the last couple of years, several protocols are proposed to enable threshold signing for well-known signature schemes such as ECDSA [19, 10, 13, 14, 8].

In this paper, we focus on one of these protocols, specifically the multiparty threshold ECDSA scheme of Gennaro and Goldfeder [13], to explain how a classical signature scheme can be converted to a multiparty threshold signature scheme.

2.1 Threshold Signatures for ECDSA

Digital Signature Algorithm (DSA) is the standard signature scheme that is adopted by NIST [18]. ECDSA is the elliptic curve variant of DSA, which is commonly used in the DLT domain. In this section, we explain how DSA, or ECDSA, can be transformed into a threshold signature scheme. Figure 1 provides an overview of the DSA.

- Given a cyclic group $\mathcal{G}$ with primer order q and a generator g, and hash functions $H : \{0,1\}^* \to Z_q$ and $H' : \mathcal{G} \to Z_q$
- **Key Generation:**
  - Private key: $x \leftarrow_R Z_q$,
  - Public key: $y = g^x \in \mathcal{G}$
- **Sign:** For an arbitrary message M,
  1. $m = H(M) \in Z_q$
  2. $k \leftarrow_R Z_q$
  3. $R = g^{k^{-1}} \in \mathcal{G}$ and $r = H'(R) \in Z_q$
  4. $s = k(m + xr) \mod q$
  5. Signature: $\sigma = (r, s)$
- **Verify:** Given the message M, the signature σ, and the public key y
  1. Validate that $r, s \in Z_q$
  2. $R' = g^{ms^{-1} \mod q} y^{rs^{-1} \mod q} \in \mathcal{G}$
  3. Iff $H'(R') = r$ accept the signature, otherwise reject.

Figure 1: Digital Signature Algorithm

In transforming a classical signature scheme to a threshold signature scheme, key generation and signing stages should be performed in a multiparty setting. The verification stage remains the same since the public key is accessible by anyone, and it can be validated by any party.

Key generation stage of the signature algorithm should be performed by all parties. The parties connect to each other and compute their own share of the private key. Value t must be known at the key generation stage, because it affects the construction of polynomials used for the generation of secrets.

Signing stage, where a signature on an arbitrary message is generated, does not require involvement of all parties. Any subgroup of size larger than t can generate a valid signature.

Due to the additive homomorphic property of secret sharing, performing additions in a threshold signature scheme is straightforward. The main challenge in transforming DSA to a threshold variant is the computation of values R and s (see Figure 1) since they require multiplication and modular inverse operations. These operations are not trivial using Shamir's secret sharing scheme. To overcome this problem, Gennaro and Goldfeder proposes a share conversion protocol (MtA) that converts multiplicative shares of a secret to its additive shares [13]. Using this protocol, Alice and Bob, holding multiplicative secret shares a and b, respectively, such that $ab = x \mod q$, can compute the corresponding additive shares α and β which results in $\alpha + \beta = x = ab \mod q$.

In Figure 2, we explain how MtA protocol works. In the protocol, E_A is an encryption of a message using Alice's private key. The encryption scheme used in the protocol is an additively homomorphic cryptosystem, such as Paillier's cryptosystem [21]. Using the homomorphic property of the encryption, Bob can compute c_B without decryption.

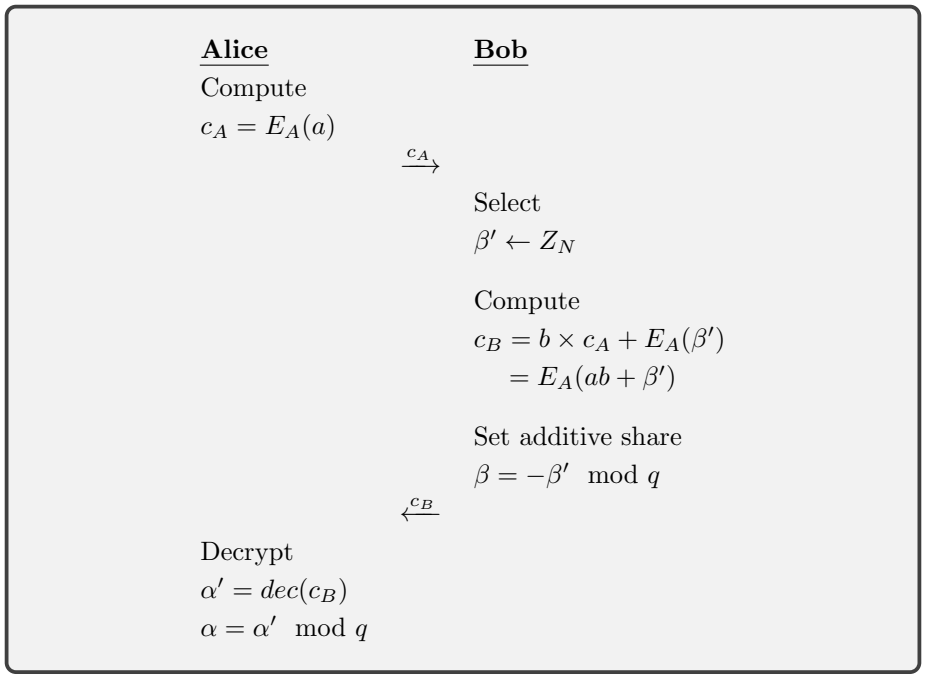

Figure 2: MtA (Multiplicative to Additive) share conversion protocol.

Particularly, Gennaro and Goldfeder's threshold ECDSA protocol works as follows:

- In the setup, parties generate shares of the secret key x that is generated as Shamir secrets with a threshold t.

- To sign a message, $t+1$ parties generate secret shares of random values k and γ.
- Using the MtA protocol (Figure 2), they compute the secret shares of the product $\delta = k\gamma$ (and reveal) and $\sigma = kx$.
- Each party multiplies their own share on γ and with the public value δ^{-1} to obtain their own share of k^{-1}.
- By computing the exponent $g^{\gamma_i \delta^{-1}}$ and combining their shares, the parties can compute $R = \prod_i g^{\gamma_i \delta^{-1}}$, which then is used to compute $r = H'(R)$.
- Finally, the secret shares of s is computed as $s_i = k_i m + w_i r$ and reconstructed as $s = \sum_i s_i$.

Once the signature (r, s) is generated with the threshold mechanism, it can be verified by any party as in the original protocol.

3 Proactive Security in Threshold Signatures

The protocol we explained in Section 2.1 is secure against malicious adversaries [13]. Similarly, other threshold signature protocols provide their security proofs regarding the underlying assumptions and adversarial model. Although a protocol is proven to be secure, its implementation might require some additional measures to protect secret data. In this section, we explain how the threshold ECDSA protocol can be extended with additional mechanisms to achieve proactive security.

Key refresh: As stated previously, in threshold cryptography a secret is distributed among n parties, and to use the secret key, $t+1$ parties must combine their shares. From an adversarial perspective, an attacker must breach $t+1$ parties to attack such a scheme. Assuming that attacker breaches the parties one by one, a key refreshing mechanism mitigates such an attack. More specifically, during the key refresh, the secret shares are refreshed after a certain period. Therefore, if the attacker succeeds to breach a percentage of parties within that certain period and breaches the remaining ones in the next period, they cannot reveal the secret, since the shares are renewed at the end of the first period. The only way for the attacker to succeed is to attack $t+1$ parties simultaneously, which is much harder.

Zero-knowledge proofs: The protocol in [13] takes an 'optimistic' approach and uses a minimum amount of zero-knowledge proofs in the protocol to obtain better performance results. One type of zero-knowledge proofs that are discussed in the paper is range proofs in the MtA protocol, which are used to prove that both Alice's and Bob's secret shares are smaller than a certain number. These range proofs are relatively expensive and impact the performance of the protocol. The authors state that removing these protocols from the protocol

might leak some information about the secret key, which is too limited and does not affect the security of the protocol. Therefore, these proofs can be included in or discarded from the implementation regarding the trade-off between security and performance.

4 Implementation details

The initial version of the protocol implementation was intended to prove the concept of ECDSA threshold signing in a multiparty setup. The first application was based on the reference implementation found in [23] and contained the proprietary network layer, the design of which was aligned with our security standards. The resulting application showed benchmarks similar to those found in [13].

The current (next) generation of the application comprises an enhanced network layer with proactive security support, as explained in Section 3, a protocol composition, parallel session support, dynamic pre-signing quorum check, and the dynamic node reconnection. The ECDSA protocol module was rewritten for this version from scratch to add range proofs (see Appendix A in [13]), exhaustive protocol error analysis, and the detection of duplicated messages. The new crypto code also addresses a few minor problems encountered by the security audit of the reference implementation[1].

It has been realized that if the protocol is wrapped in a general-purpose finite state machine (FSM), that improves the usability and maintainability of the whole application. With the addition of FSM, the library can now facilitate virtually any multiparty interactive protocol. Finally, we added the abstract storage level API to accommodate various types of secret vaults.

The choice of the implementation language: Rust is a relatively young programming language. The young age of the language reflects on the ecosystem stability and the overall language accessibility due to partially lacking documentation, poor debugging support (as compared to more mature C/C++/C#/Java). Despite that, the language has numerous advantages. Due to its strict rules of memory management, Rust delivers safe code. Rust compiler produces efficient output optimized for performance while preserving the ability to access the data at a very low level, similar to what C/C++ does. Next to it, Rust facilitates foreign function interface so that programs written in other languages can call Rust dynamic library modules via standard C-style bindings. We have chosen this language for a few reasons:

- The language has aforementioned properties which make it suitable for an application in cryptography in general;
- The initial version of the code required bindings with the reference implementation of the protocol written in RUST;

[1] `https://github.com/ZenGo-X/multi-party-ecdsa/tree/master/audits`

- It supports functional programming style so that algorithms are easier to validate;
- The language gains popularity.

Acknowledgements

The authors thank Victor Ermolaev, Tommy Koens, Shariff Lutfi, and Antoine Girard for their valuable feedback in the preparation of this document.

References

[1] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[3] Azer Bestavros, Andrei Lapets, and Mayank Varia. User-centric distributed solutions for privacy-preserving analytics. *Commun. ACM*, 60(2):37–39, 2017.

[4] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *Financial Cryptography*, volume 8975 of *Lecture Notes in Computer Science*, pages 227–234. Springer, 2015.

[5] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[6] Luis T. A. N. Brandao, Michael Davidson, and Apostol Vassilev. Nist roadmap toward criteria for threshold schemes for cryptographic primitives. `https://csrc.nist.gov/publications/detail/nistir/8214a/final`, 2020.

[7] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002.

[8] Ran Canetti, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA. *IACR Cryptol. ePrint Arch.*, 2020:492, 2020.

[9] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19. ACM, 1988.

[10] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE, 2019.

[11] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Found. Trends Priv. Secur.*, 2(2-3):70–246, 2018.

[12] Brian Gallagher, Darwin Lo, Peter Frands Frandsen, Jesper Buus Nielsen, and Kurt Nielsen. Insights network - a blockchain data exchange. `https://s3.amazonaws.com/insightsnetwork/InsightsNetworkWhitepaperV0.5.pdf`, 2017.

[13] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. *IACR Cryptol. ePrint Arch.*, 2019:114, 2019.

[14] Rosario Gennaro and Steven Goldfeder. One round threshold ECDSA with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020:540, 2020.

[15] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[16] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[17] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[18] David W Kravitz. Digital signature algorithm, 1993. US Patent 5,231,668.

[19] Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO (2)*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.

[20] Yehuda Lindell. Secure multiparty computation (MPC). *IACR Cryptol. ePrint Arch.*, 2020:300, 2020.

[21] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[22] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[23] ZenGo X. Multi-party ecdsa. `https://github.com/ZenGo-X/multi-party-ecdsa`, 2020.

[24] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.