

# Threshold Signatures for Central Bank Digital Currencies

Mostafa Abdelrahman<sup>1,2</sup>, Filip Rezabek<sup>1</sup>[0000–0002–9090–5633],  
Lars Hupe<sup>1,2</sup>✉[0000–0002–8442–856X], Kilian Glas<sup>1</sup>, and Georg Carle<sup>1</sup>

<sup>1</sup> Technische Universität München, Munich, Germany

<sup>2</sup> Giesecke+Devrient, Munich, Germany

✉ lars.hupe@tum.de

**Abstract.** Digital signatures are crucial for securing Central Bank Digital Currencies (CBDCs) transactions. Like most forms of digital currencies, CBDC solutions rely on signatures for transaction authenticity and integrity, leading to major issues in the case of private key compromise. Our work explores threshold signature schemes (TSSs) in the context of CBDCs. TSSs allow distributed key management and signing, reducing the risk of a compromised key. We analyze CBDC-specific requirements, considering the applicability of TSSs, and use Filia CBDC solution as a base for a detailed evaluation. As most of the current solutions rely on ECDSA for compatibility, we focus on ECDSA-based TSSs and their supporting libraries. Our performance evaluation measured the computational and communication complexity across key processes, as well as the throughput and latency of end-to-end transactions. The results confirm that TSS can enhance the security of CBDC implementations while maintaining acceptable performance for real-world deployments.

**Keywords:** Threshold Signatures · ECDSA · CBDC

## 1 Introduction

Digital signatures are essential to ensure the authenticity and integrity of online data. They provide a way to verify the origin and integrity of a message, ensuring that the message has not been tampered with and that it indeed comes from the claimed sender. The system’s security collapses if the private key  $sk$  is compromised, as an attacker could forge signatures. Conversely, if the  $sk$  is lost or destroyed, valid signatures cannot be created, leading to availability loss. This overall leads to a single point of failure.

Threshold Signatures Schemes (TSSs) address the vulnerabilities associated with a single point of failure in private key management. The TSS distributes the signing authority among multiple parties, requiring a subset (or threshold  $t$ ) of these parties to collaborate to produce a valid signature. This trust distribution reduces the risk of key compromise and increases the resilience of the

---

Mostafa Abdelrahman and Filip Rezabek contributed equally to this paper.

cryptographic system. By eliminating a single point of failure, TSSs are particularly advantageous in high-stakes applications such as financial transactions and secure communications.

A key application of TSSs is within blockchain and digital currencies, which require robust security measures due to the significant value they protect [23]. This is particularly relevant in the context of Central Bank Digital Currencies (CBDCs), which are digital representations of a nation’s currency issued and regulated by the central bank. CBDC introduces the role of a Financial Service Provider (FSP), which process users’ payments and ensure funds availability. In that case, the FSP manages the private key of users as custodial key management. Naturally, a common design pattern uses online wallets distributed among various FSPs. However, this puts a lot of trust in the FSPs to create and manage hosted wallets. Besides, the FSPs do not trust each other. Therefore, TSS is a viable solution as the threshold number of nodes within a FSP have to collaborate, increasing the resilience of the FSP system. In general, we examine the integration of TSS in the context of CBDC deployments. For the scope of this paper, we select the Filia implementation [30], an off-chain CBDC solution developed by Giesecke+Devrient (G+D). Filia follows a similar design with a focus on FSPs for custodial key management, allowing the generalization of our findings to other CBDC implementations with similar architecture.

Filia offers hardware-based (offline) and hosted (online) wallets. Applying threshold signatures to hardware wallets would be cumbersome for day-to-day transactions since it would require the user to juggle multiple devices. Filia’s online wallets currently only support Elliptic Curve Digital Signature Algorithm (ECDSA). For that reason, we focus on State-of-the-Art (SotA) ECDSA TSS as they apply to online wallets. The goal is to develop a threshold version of this key management system to enhance security by preventing single-point failure.

*Contributions* We provide these key contributions:

- Analysis of SotA TSS based on ECDSA;
- Protocol design fitting the CBDC use cases;
- Theoretical and empirical evaluation of the protocol.

## 2 Background

This section provides an overview of the CGGMP21 ECDSA TSS, Filia as the selected case study CBDC, and Environment for Generic In-vehicular Networking Experiments (EnGINE)/Multilayer Environment and Toolchain for Holistic NetwOrk Design and Analysis (METHODA) framework used for evaluation.

*CGGMP21* The CGGMP21 protocol is a SotA  $n$ -out-of- $n$  threshold ECDSA scheme [14]. It provides features fitting the context of CBDC applications. First, it relies on Paillier encryption for secure multiplication of secret shares and utilizes Zero-Knowledge Proofs (ZKPs) to detect and attribute deviations from

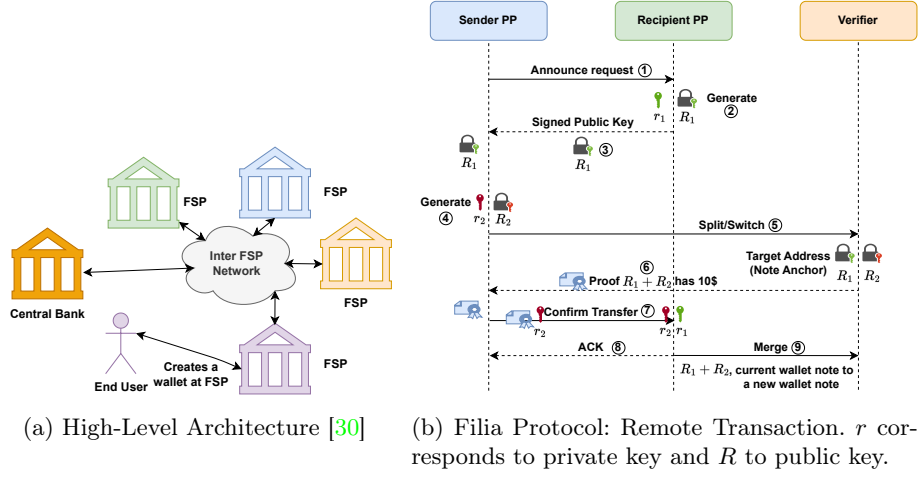


Fig. 1: Filia Architecture and Protocol Overview. Figures outline the involved parties and their form of interaction. Color differentiates between the parties.

the expected behavior [32, 45]. Next, it supports pre-signing, identifiable aborts, security against adaptive adversaries, and universal composability under strong RSA, DDH, and Paillier security assumptions. It provides proactive security with periodic key refreshes and ensures that compromised shares within an epoch do not affect the system’s integrity. It offers efficiency trade-offs: a 3-round presign protocol with  $O(n^2)$  identification cost or a 6-round presign protocol with  $O(n)$  identification cost. The protocol can be extended to general  $t$ -out-of- $n$  threshold case using Shamir’s secret sharing (SSS) and converting Shamir secret shares into additive shares [55, 18].

*Filia* Figure 1a provides a high-level overview of the Filia system. The central bank is responsible for the core infrastructure. At the same time, private sector intermediaries, known as FSPs, offer end-user wallets and develop services based on the technology and regulatory framework established by the central bank. Each FSP maintains a wallet at the central bank, where the requested liquidity is transferred. Other CBDC solutions with similar architecture exist.

*EnGINE/METHODA* The EnGINE/METHODA framework [48, 49], initially developed for Time Sensitive Networking (TSN) emulations, provides a robust and scalable environment designed to support reproducible and modular experiments in distributed cryptographic operations. Built with Ansible-based orchestration, METHODODA allows for flexible deployment and management of complex setups, making it well-suited for evaluating cryptographic protocols in distributed systems.

### 3 Related Work

Existing research on CBDC system design highlights different models, including two-tier architectures where central banks issue digital currency but rely on intermediaries for distribution [7] which Filia follows. However, security remains a critical challenge in CBDC implementations, particularly in mitigating operational and cyber risks [17]. This paper addresses these concerns by exploring TSS as a means to enhance key management security and reduce the risk of private key compromise.

There is limited work in applying TSSs in the context of CBDC, with existing research focusing primarily on distributed ledger-based implementations. PEReDi [53] employs TSS in a scenario where maintainers, including banks, financial institutions, and regulatory offices, each hold a share. This setup allows for selective transaction tracing and privacy revocation in cases of suspicious activities. IBM’s CBDC framework [5] incorporates TSS within its new endorser model, where TSSs enhance security and accountability by requiring a collective agreement from multiple endorsers for transaction validation. Overall, the framework focuses on hardening the token creation by the CBDC, minimizing the likelihood of the creation of unauthorized tokens. On the other hand, our threat model aims to protect against theft or unauthorized use of tokens.

Numerous initiatives within the cryptocurrency industry, many of which are part of the MPC Alliance [43], have adopted TSSs to secure digital assets. This distributed approach has enabled secure “Wallet as a Service” models [53, 59, 26]. These platforms enhance security by distributing cryptographic operations across multiple parties, ensuring robust protection even when certain parties may be compromised.

## 4 Analysis & Protocol Design

The main CBDC use case analyzed in this work considers the FSP as the custodian of users’ wallet private keys. In this model, the FSP securely manages private keys on behalf of multiple users and an entrusted central ledger maintains a record of public keys to ensure the legitimacy of the money. This section focuses on the private key lifecycle and its distribution relying on the TSS. First, we provide an overview of the Filia platform and possible solutions on how TSS can be integrated into it. We define general requirements relevant to the context of CBDC along with a detailed security model. We argue for selecting a suitable TSS based on the requirements.

### 4.1 Filia Protocol

Filia relies on a modified unspent Transaction Output (UTXO)-based model used to represent a digital currency. Each UTXO, referred to here as a *note*, consists of a public-private key pair  $(r, R)$  derived from the secp256r1/P-256 curve and a denomination  $(v)$  indicating its monetary value. The private component  $r$  is

securely managed within the hosted wallet by the *Payment Processor (PP)*, the main component of the FSP responsible for wallet management and transaction handling. Meanwhile, the public component  $R$  is registered and tracked by a Verifier, a central bank infrastructure tasked with preventing double spending and ensuring system integrity.

Transactions in this system follow specific rules. Certain transaction types, such as currency issuance or destruction, are restricted to the central bank, while old transaction outputs are periodically garbage collected to improve system efficiency. Transaction commands involve consuming one or more existing notes and creating new ones, with the available commands being **Create**, **Split**, **Merge**, **Switch**, and **Destroy**. Each transaction is validated by the *Verifier*, which ensures the correctness of commands and prevents double-spending. Notes are registered in the system only if they are unique, and commands are rejected if any input notes have already been consumed in a prior transaction.

The Filia system differentiates between two types of transfers: **local transfers** (wallets within the same FSP) and **remote transfers** (wallets at different FSP). While local transfers follow a simplified protocol, remote transfers require additional security measures due to the lack of trust between different FSP. A non-repudiation protocol addresses this, ensuring that the sending PP can provide proof of the money sent. Figure 1b illustrates the steps involved in a remote transfer under the Filia protocol:

1. **Anounce Transfer:** The recipient PP receives a announcement request from the sender PP with value  $(v)$  ①. The recipient PP generates a key pair  $(r_1, R_1)$  ② and sends the public key  $R_1$  back to the sender PP ③.
2. **Register New Public Key:** The sender PP generates a second key pair  $(r_2, R_2)$  ④ and sends a request to the verifier to confirm the transfer and register the value  $(v)$  under  $R_1 + R_2$  ⑤.
3. **Store Verifier Proof ⑥:** The sender PP stores a proof provided by the Verifier, demonstrating that the value  $(v)$  has been transferred to the correct address  $R_1 + R_2$  at the Verifier.
4. **Confirm Transfer:** The sender PP sends the private key  $r_2$  to the recipient PP ⑦, allowing the recipient to access the funds and the recipient PP replies with an acknowledgment ⑧.

Eventually, only the recipient PP knows  $r_1 + r_2$  and can use the note. The recipient PP can switch the funds to a new key or merge the funds to a previously available note ⑨.

This process ensures that the recipient cannot access the transferred funds until the sender has securely stored the proof of transfer. Additionally, because the sender does not know the private key  $r_1$ , they cannot reclaim the money after it has been sent.

Should the sender PP not disclose  $r_2$ , the value  $v$  will be locked; however, it will also be unable to prove correct transfer execution.

## 4.2 Possible Solutions

Continuing with the protocol overview, we explore two potential solutions for implementing threshold key management within the Filia system at the FSP level. We outline and analyze each solution, including its unique pros and cons.

**Solution 1: Payment Processor-Based Threshold Key Management** In this approach, each FSP deploys  $n$  PPs, with each PP holding a share of the private key. Specifically, each PP holds a new note in the form  $(v, r_1)$ ,  $(v, r_2)$ , ...,  $(v, r_n)$ . Here, the TSS is directly implemented within the payment processing system, so when a transaction requires signing, a predefined number of PP collaborate to generate the signature.

**Solution 2: Separate Key Management Network (KMN)** This solution establishes a dedicated KMN within the FSP infrastructure, where a set of  $n$  specialized nodes (key management nodes) are responsible for generating and storing key shares, and providing signatures. PP sends requests to the KMN for signatures or key generation as needed. Each node in the KMN holds a key share  $r_1, r_2, \dots, r_n$ , and assigns a specific identifier (uuid) to the shares, which is sent to the PP. Consequently, the PP holds a **note** in the form  $(v, \text{uuid})$  and uses that uuid to request a signature from the KMN.

**Discussion** When considering the complexity, **Solution 1** tightly integrates key management into the payment processing infrastructure, requiring each PP to maintain its database of key shares. This necessitates synchronization across multiple PPs, significantly increasing the operational overhead, particularly in achieving consensus between PPs. In contrast, **Solution 2** isolates key management in a dedicated network, simplifying the responsibilities of PPs. Although this reduces complexity within the PPs, it adds a layer of infrastructure that requires careful management, such as ensuring the availability and reliability of the KMN nodes.

Regarding latency, **Solution 1** has an advantage due to the proximity of all operations within the FSP infrastructure. On the other hand, **Solution 2** introduces communication overhead as PPs need to interact with the separate key management nodes. This added step can lead to higher latency, particularly in high-frequency transaction environments.

The modular design of **Solution 2** ensures that the TSS can be updated or replaced without requiring significant changes to the PP codebase. The system can also take advantage of batching and parallelization to reduce delays and improve efficiency without disrupting other PP tasks. By decoupling key management from the payment processing system, KMN enhances security, as keys remain secure within the storage of dedicated nodes, even if PP is compromised.

The KMN is selected for its modular design, enhanced security, and ease of updates. Decoupling key management from payment processing eliminates the

need for consensus mechanisms required in **Solution 1**, reducing the implementation complexity of Proof-of-Concept (PoC). Furthermore, KMN enables efficient batch processing and parallelization, ensuring scalability without disrupting payment processing tasks, making it a robust choice for the Filia system.

### 4.3 System Requirements

We outline several key requirements for extending the Filia protocol by TSS and considering **Solution 2**. They must easily migrate Filia wallets and transaction signing to the TSS system. Notably, these requirements should be generalizable to designs similar to CBDC as Filia.

**R1 - Transparent Support for Standard and Threshold Wallets:** The system must conceal the type of wallet in use, whether standard or threshold, from external observers, including other FSPs. This ensures flexibility within the Filia system, enabling different security levels based on factors such as the wallet's balance.

**R2 - Efficient Distributed Key Generation (DKG):** Due to multiple generation of key during transactions in Filia, the DKG process must be optimized for high efficiency.

**R3 - High Signature Throughput:** The signing process must achieve high throughput to handle the multiple signatures required per transaction and efficiently load high user loads. For instance, during a remote transfer, the process involves splitting the sender's wallet and merging the transferred value into the recipient's wallet (as shown in Figure 1b, Steps ⑤ and ⑨). These operations require multiple signatures, so the system must be capable of processing these quickly.

**R4 - Secure Key Reconstruction:** The system must enable the secure reconstruction of short-lived keys when required. Specifically, when the sender's PP generates a threshold key pair  $(r_2, R_2)$ , it must securely reconstruct or export  $r_2$  during the process of transmitting the full key to the recipient's PP (e.g., Figure 1b, Step ⑦).

**R5 - Key Update/Key Addition:** The system must facilitate the secure addition of new key parts to existing key shares. For instance, as illustrated in Figure 1b, the sender PP must send  $r_2$ , which is a full key, to the recipient PP. The recipient PP then combines  $r_2$  with their threshold key  $r_1$ . The result is that each node holds a share of the combined key  $r_1 + r_2$ .

**R6 - Compatibility with Existing Infrastructure:** The TSS must be fully compatible with ECDSA, in specific secp256r1/P-256 curve, to align with the existing infrastructure, particularly for hardware wallets that rely on smart card technology that use ECDSA. This requirement ensures that the integration of TSSs does not disrupt existing systems and maintains consistent security across all wallet types.

**R7 - Security:** The TSS should be secure with minimum assumptions and should be secure even when combined with other systems/protocols.

#### 4.4 Security System Model

We define the environment and conditions under which the protocol is expected to operate.

**Adversary Model** In our system, we assume the presence of an active, malicious adversary. This adversary can arbitrarily deviate from the protocol, aiming to disrupt the system or gain unauthorized access to information.

**Threshold Structure:** We operate under the assumption of a dishonest majority ( $t < n$ ), as the protocols provide stronger security guarantees against active adversaries. Even though honest majority protocols might seem appropriate given that all nodes are deployed at the FSP level by a single entity, we choose dishonest majority due to stronger security guarantees against active adversaries. However, this comes at the cost of robustness, which cannot be achieved under dishonest majority settings [24, 16].

**Computational Power:** Given the active security setting with a dishonest majority, the protocol should aim to achieve computational security as perfect security is unattainable under these settings [24].

**Corruption Power:** This model assumes static corruptions, as frequent key changes in Filia eliminate the need for proactive security or handling mobile adversaries. Key updates occur naturally during transactions, making explicit key refreshing after specific epochs unnecessary. However, this assumption can be extended to include proactive security measures for wallets that remain inactive for extended periods.

**Network and Communication Model** Our system assumes all signatories are connected through an authenticated and synchronous broadcast mechanism within a single FSP. This setup ensures that communication is public, and any message sent will either be received by all participants in the next round or not be delivered at all. Without pre-established authenticated communication, an adversary who previously controlled a corrupted party and managed the communication channels can impersonate that party indefinitely [12].

The synchronous broadcast mechanism is crucial for maintaining accountability and ensuring all parties reach a consensus. Without defined communication delays, holding any signatory accountable for failing to respond would be impossible. Additionally, reliable broadcast is vital for distributing proofs during the DKG process and upholding accountability throughout the protocol [14].

#### 4.5 Selection of Threshold ECDSA Scheme

TSSs have been extensively explored for their role in securing digital assets. Table 2 compares some popular libraries available and implementations of TSSs.

Two comprehensive surveys [6, 54] provide an overview of key developments in the field. SotA protocols such as CGGMP21 [14] and DKLS23 [22] build on earlier advancements [13, 28, 33, 20, 21]. These protocols follow standard steps:



Scheme	Robust	Corruption Assumptions	Strategy
Honest Majority			
Gennaro et al. [29]	✓	Static	None beyond unforgeability of DSS signatures
Damgård et al. [19]	✓	Static	None beyond ECDSA security
Pettit [46]	✓	Static	Decisional Diffie–Hellman (DDH), ECDSA unforgeability
Bonte et al. [10]	✗	Static	Discrete Logarithm Problem (DLP), MPC-friendly hash function
Shi et al. [56]	✓	Static	None beyond EdDSA security
Stinson and Stroh [57]	✓	Static	DLP
Komlo and Goldberg [36]	✓	Static	DLP, Secure hash function
Kondi and Ravi [38]	✗	Static	Public Key Infrastructure (PKI)
Jonathan Katz and Antoine Urban [34]	✗	Static	Semi-honest coordinator, unforgeability of standard ECDSA
Dishonest Majority			
Canetti et al. [14]	✗	Adaptive	Strong-RSA, ECDSA unforgeability, semantic security of Paillier encryption, DDH
Abram et al. [3]	✗	Static	Ring-LPN, ECDSA unforgeability
Doerner et al. [22]	✗	Static	Ideal commitment and two-party multiplication primitives
Feng et al. [25]	✗	Adaptive	LPN, PCF
Komlo and Goldberg [37]	✗	Static	DLP
Ruffing et al. [50]	✓	Adaptive	DLP
Lindell [40]	✗	Static	Ideal commitment and PKI

Table 1: Comparison of Threshold ECDSA, EdDSA, and Schnorr Signature Schemes. ✓ indicates that the property is achieved, while ✗ indicates that it is not

- Rewriting the ECDSA signing equation into an “Multiparty Computation (MPC)-friendly” equivalent.
- Employing cryptographic primitives for secure multiplication (e.g., Oblivious Transfer, Paillier, or Class Groups).
- Verifying that all operations are performed honestly.

The primary distinctions between these protocols lie in their security definitions, underlying assumptions, and communication and computation complexities, which depend on the used cryptographic primitives. Table 1 provides a summary of recent threshold signature schemes, detailing key differences in threshold settings, robustness, corruption strategies, and security assumptions.

Among the libraries evaluated, the implementation of CGGMP21 by *Lockness* [41] meets our key system requirements. First, unlike other libraries, it is production code, not a PoC such as *silence-labs* [51]. Next, the library supports *secp256r1*, the elliptic curve that *Filia* (R6) requires. Additionally, it enables essential functionalities such as importing existing keys (R5) and exporting keys

Library	Curves	Key Export/ Import	Audited	Lang.	Docs	Purpose
Honest Majority						
Block-Daemon [8]	P-224/256/384/521, Ed25519, Ed448	✓	✓	Go, Java, Node	+++	Prod.
Nakasendo [44]		✗	✗	C++, Python	+	Undefined
Dishonest Majority						
Lockness [41]	secp256r1, stark	✓	✓	Rust	++	Prod.
circlefin [15]	ed25519	✗	✗	Go	+	Prod.
silence-labs [51]		✗	✗	Rust	+	PoC
OxCarbon [2]		✗	✗	Rust	+	Undefined
ZenGo-X [1]		✗	✓	Rust	++	PoC
alephzero [4]		✗	✗	Go	-	PoC
taurusq-io [58]		✗	✓	Go	+	Undefined
cait-sith [42]		✗	✗	Rust	+	PoC
bnb-chain [9]	Edwards	✗	✓	Go	++	Prod.
Protect [47]		✗	✗	Java	+++	PoC
Safe-heron [52]	P-256, stark	✗	✓	C++	+	Prod.
Framework						
MP-SPDZ [35]	P-256	✓	?	Python, C++	+++	PoC

Table 2: Comparison of software libraries of TSSs. All support at least secp256k1, so we do not list it separately. The *Docs* column indicates the quality of documentation: more “+” or “-” for better or poorer documentation, respectively. ✓ indicates the feature is supported

securely from nodes (**R4**). One of the key advantages of CGGMP21 is its efficient signing process, which is crucial for achieving high throughput in the Filia system. The protocol employs pre-signing and a one-round online signing process. This allows most cryptographic work to be completed ahead of time, minimizing delays during transactions and directly addressing the system’s need for high signature throughput (**R3**). CGGMP21 is a SotA threshold ECDSA scheme with four signing rounds.

The protocol’s security is further enhanced by its Universal Composability (UC) [11], ensuring that CGGMP21 remains secure even when composed with other cryptographic protocols. The UC-security guarantee fulfills the requirement for strong security (**R7**). Thus, the combination of compatibility, efficiency, and security makes CGGMP21 an optimal choice for the Filia system, particularly with the *Lockness* [41] library.

## 5 Theoretical and Empirical Evaluations

The section provides an overview of the experiment design and the evaluation findings.

### 5.1 Security Analysis

As a part of the theoretical evaluation, we consider the communication and computational complexities of the CGGMP21 protocol. This information provides details about the scalability of our solution, which is a critical consideration for its applicability. Table 3 summarizes the complexities from the perspective of a single node, including the respective communication rounds and Big(O) cost.

The security of the KMN falls back on the security of the CGGMP21 protocol, as implemented in the *Lockness* library, which has been rigorously audited. However, additional considerations for KMN security include the security of performing background DKG and key assignments. The execution of DKG in the background does not compromise overall security, as unauthorized access to the database of a node would expose the key shares of that node, regardless of when or how they are generated. Moreover, considering the system architecture, including the PP, an adversary who gains control of the PP would not be able to forge signatures. This is because every transaction request to the PP requires an authentication token tied to the user, such as biometric verification stored securely on the user device enclave. While an attacker in this scenario could disrupt operations, such activity would be detectable, allowing for a prompt restart of the PP to restore functionality. Notably, a coordinator is introduced in the implementation because it simplifies communication to facilitate the implementation of PoC.

The *Lockness* library operates under two primary assumptions: (1) all messages are authenticated, and (2) all Peer-to-Peer (P2P) messages are encrypted. These requirements are fulfilled using the *libp2p* library, which ensures secure and encrypted communication channels. Additionally, the library assumes reliable broadcast, achieved using an integrated reliability check mechanism. This mechanism enables each participant to hash the messages received in the previous round and include these hashes in their outgoing messages. By comparing these hashes, participants verify that all parties have received identical messages, ensuring consistency and reliability throughout the protocol execution.

### 5.2 Experiment Design & Empirical Evaluation

The implementation is based on the Rust-based *Lockness* library [41]. The library abstracts the network layer, requiring only stream-based input and output for message exchange. For this, we rely on the *libp2p* library [39] to handle peer-to-peer communication.

We adopt the KMN approach (i.e., **Solution 2**). In this setup, each FSP integrates its KMN. Keys and pre-signatures are generated in advance by the KMN nodes. When a PP sends a key generation request, the KMN assigns an unallocated key to the request. Pre-signatures are utilized for signature operations on Filia commands; otherwise, the system falls back to an interactive signing step where the PP waits for the KMN nodes to interactively generate a signature. We introduce a coordinator node between the client PP and the KMN nodes. This node collects key shares from the KMN nodes to reconstruct the full

Phase	Round	Computation	Communication
<b>Key Generation</b>	Round 1	$O(1)$	$O(n)$
	Round 2	$O(1)$	$O(n)$
	Round 3	$O(n)$	$O(n)$
	Output	$O(n)$	no communication
<b>Presigning</b>	Round 1	$O(t)$	$O(t)$
	Round 2	$O(t)$	$O(t)$
	Round 3	$O(t)$	$O(t)$
	Output	$O(t)$	no communication
<b>Signing</b>	Round 1	$O(1)$	$O(t)$
	Output	$O(t)$	no communication

Table 3: Complexities for CGGMP21 protocol [14, 31]

key during key export. It also manages key import operations by receiving a full key and splitting it into  $n$  key shares for the key addition/update process. Additionally, the coordinator node aggregates pre-signatures (also known as partial signatures) from the KMN nodes to produce the final signature, which is then sent back to the client.

The evaluation is conducted in two phases. First, we benchmark cryptographic operations independently to analyze performance under varying  $n$  and  $t$ . These should provide insights into the cost following the overview in Table 3. Second, we integrated the updated protocol into the Filia system and measured End-to-End (E2E) transaction performance, including latency and throughput.

The cryptographic operations are evaluated in a local testbed using the METHODA [48, 49] framework. Our experiments run on four dedicated nodes. Each dedicated node has 64 virtual CPUs (vCPUs) and 768 GB of RAM. Each experiment includes application code and service management files, which handle configuration and execution across nodes. Within METHODA, scenarios define sequences of experiments, allowing controlled adjustments to parameters like  $n$  and  $t$ . The nodes are interconnected over a dedicated test switch. To assess the scalability of the solution, we carry out experiments with various values of  $t$  and  $n$  from  $t = 2, n = 3$  to  $t = 32, n = 52$ . We evaluate individual steps, including DKG, pre-signing, and signing. Each experiment involves 100 iterations and uses a single thread per node. These parameters are chosen based on evaluations of other threshold signature schemes (e.g., FROST [37], GG18 [27], and GG20 [28]) and relevance to expect the scale of the CBDC deployments. We fix the message size to 256 bit. We measured the computation and I/O time for each step during the experiment run. The **computation time** is the time each node spends on local cryptographic operations. On the other hand, the **I/O time** is the total time each node waits for messages to proceed to the next round. Therefore, the total iteration time is the sum of computation and I/O times. This evaluation uses the PerfProfiler utility from the *Lockness* library. Time measurement begins at the start of the iteration once the room ID is assigned and all nodes are ready

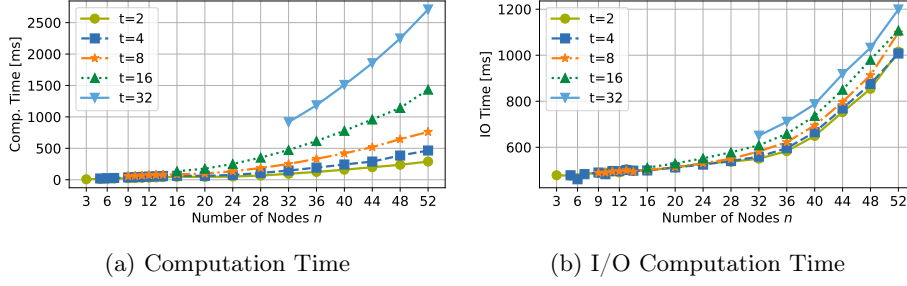


Fig. 2: DKG Time vs Number of Nodes

to execute the protocol. This assumes all expected nodes can communicate with each other.

Each epoch begins when all nodes in the protocol discover and authenticate each other. Each node waits for a defined number of nodes  $n$  and then assigns each protocol iteration a unique room ID with a dedicated communication channel for that iteration's messages. Time measurement begins at the start of the iteration once the room ID is assigned and all nodes are ready to execute the protocol.

**Results of Phase 1** First, we conduct experiments of the DKG as shown in Figure 2. The results show that increasing the number of nodes and the threshold directly impacts the time required for DKG. As shown in Figure 2a, the time increases quadratically with the number of nodes and threshold. This is also the case for the I/O time when each node has to wait to proceed, as shown in Figure 2b. We observe that for a smaller number of nodes, the I/O time exceeds computation time, likely due to communication delay among nodes. However, computation time begins to dominate the total DKG duration as the number of nodes and threshold increase, observing computation time up to 2500 ms for  $n = 52$  and  $t = 32$ . More time is spent on computation than on I/O.

As pre-signing and signing are not dependent on  $n$ , we average all values for respective computation and I/O times. As shown in Figure 3, the pre-signing follows a similar tendency for increasing  $t$ . Figure 3a shows computation time with a quadratic increase. For the I/O time, we observe fractional cost in comparison to the computation time, as shown in Figure 3b. Continuing with signing presented in Figure 4 we observe some order of times as for pre-signing. Especially the Figure 4a is almost identical to the Figure 3a. However, for I/O time we observe an increase, as introduced in Figure 4b. The time difference between pre-signing and signing is minimal, indicating that the final online round, which produces the signature on the known message, is computationally inexpensive.

**Results of Phase 2** The integration testing is carried out locally on a MacBook M3 Pro due to the complexity of setting up a private cloud-controlled

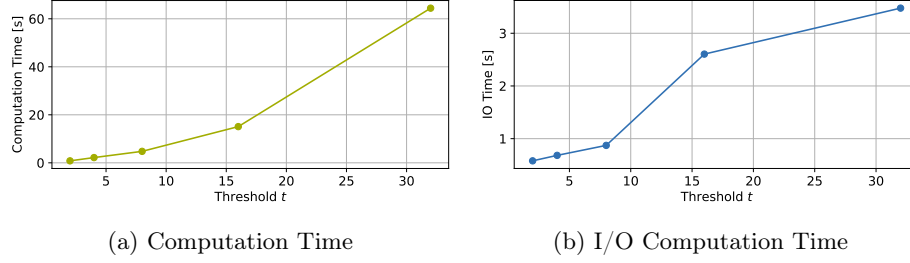


Fig. 3: Pre-signing Time vs Threshold

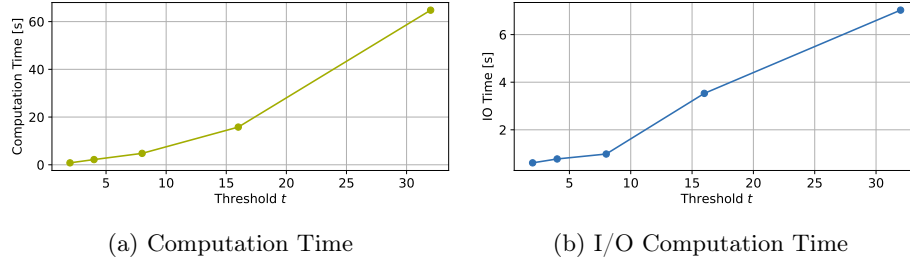
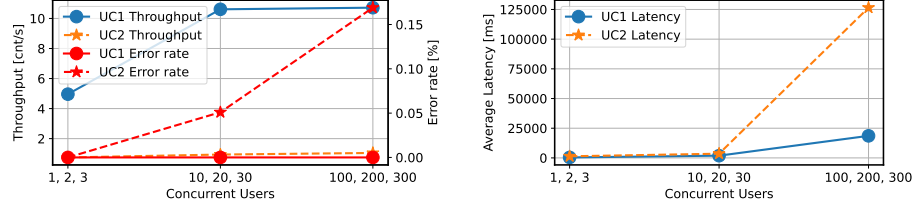


Fig. 4: Signing Time vs Threshold

environment and the challenges of porting the code to external infrastructure. The setup includes two KMNs ( $t = 2, n = 3$ ), two FSPs hosting PPs, and a central Verifier hosted in a Docker container to validate transactions. We test two use cases (UCs): UC1 transfers between customers of the same FSP, whereas UC2 emulates transfers between different FSPs. The evaluation simulated varying loads with 1, 2, 3, 10, 20, 30, 100, 200, and 300 concurrent users over 30-minute test sessions. Wallets were pre-created using APIs exposed by the PPs, with each wallet assigned sufficient funds to execute transfers. The load tests measured E2E transaction timing, starting from the API call to initiate the transfer until the recipient’s wallet confirmed transaction completion. As shown in Figure 5a, throughput increases steadily as the load increases in UC1, while in UC2, throughput improves slightly. In both cases, the latency increases as the load increases, as shown in Figures 5a and 5b. Performance evaluations reveal an order-of-magnitude decrease in throughput when using threshold signatures, particularly for cross-FSP transfers. Under high loads (100, 200, and 300 concurrent users), UC2 throughput drops to approximately 1.2 Transactions Per Second (TPS), compared to 60 TPS in a non-threshold configuration. Similarly, UC1 throughput decreases from 190 to around 10.7 TPS in the same setup. We also observe failed transactions with more concurrent users for throughput with an error rate of up to 0.15%, as shown in Figure 5a.



(a) Total throughput w/ increasing concurrent users (b) E2E latency with increasing concurrent users

Fig. 5: E2E transactions latency and throughput

### 5.3 Discussion

The CGGMP21 protocol and the KMN architecture collectively address all the requirements described, making them a fitting choice for Filia and other CBDC systems. However, further research is necessary to address performance challenges with threshold cryptography. As the number of nodes and the threshold increase, the performance of the cryptographic operations degrades noticeably. This is due to the growing computational and communication complexity, which scales with  $n$  and  $t$ , as detailed in Table 3. In practical deployments, a smaller number of nodes is often sufficient to maintain better performance, and unless operated properly, larger node counts do not necessarily yield more security advantages. It is important to note that integration testing results are based on executions in a testing environment; a speedup is expected in a production-ready setup, provided the environment is carefully tuned to optimize performance and minimize resource constraints. While the added latency of communication between the PP and the KMN introduces a slight overhead, the system’s scalability is mainly influenced by bottlenecks identified during the PoC.

A key bottleneck appears in the time spent during signing operations without a pre-signature. This delay is introduced during the key import and update step, where the new key  $r_1 + r_2$  has no pre-signature (⑦ in Figure 1b). Consequently, a complete interactive signing on the KMN side is required when merging and signing with this key. Sometimes, it is triggered by an interactive signing or key generation. This happens when the pre-generated keys are exhausted, or a key exists without a pre-signature. Therefore, the nodes have to re-initialize communication and authentication with each other, leading to higher latency and resource inefficiencies. Addressing these bottlenecks may involve triggering a pre-signing operation in the background upon key updates. Similarly, nodes can establish communication channels among different protocol executions to reduce the overhead of re-establishing the secure channel between nodes.

The results for UC2 further indicate a non-zero error rate, with transaction timeouts caused by resource constraints. Transactions exceeding a 5-minute processing time are classified as errors, often due to unhandled temporary issues or resource locking by other jobs. Notably, an error rate is also observed in

non-threshold transactions, indicating that the issue is not solely attributable to the use of TSS but rather to resource availability and system load. A potential solution to mitigate these errors would be horizontal scaling, allowing the system to dynamically allocate additional resources as transaction loads increase. In contrast, UC1 demonstrates zero errors, reflecting the lower complexity of same-FSP transfers compared to cross-FSP transfers.

The experimental design of this evaluation follows a logical progression that validates the solution from PoC to performance assessment. The E2E transaction evaluations reveal the impact of TSSs on performance under various load conditions, replicating real-world transaction demands. Isolating cryptographic operations allows a detailed examination of resource costs associated with DKG, pre-signing, and signing. Although DKG and pre-signing occur as background tasks, their isolated evaluation highlights the computational demands they introduce. The signing evaluation shows the effect of the key update bottleneck, which requires interactive signing at that step. A configuration of  $t = 2$  and  $n = 3$  requires approximately one second for an interactive signing process. While manageable under a small load, this latency represents a substantial performance cost as the system scales.

## 6 Conclusion

Our work improves the security and resilience of CBDC systems, using Filia as a representative platform to address typical requirements. A comprehensive theoretical analysis identifies key system requirements, including compatibility with existing infrastructures, high signature throughput, secure key reconstruction, key updates, and strong security guarantees. We also define the computational and communication complexity of the CGGMP21 protocol and establish a robust security model to address adversarial behaviors. Building on this foundation, the CGGMP21 protocol is selected and used to build a PoC, along with the design and implementation of a KMN architecture. Empirical evaluation shows that the evaluation of cryptographic operations aligns with the complexity analysis. Moreover, E2E transaction performance decreases by an order of magnitude compared to non-threshold systems, particularly in cross-FSP transactions. Despite this, the system demonstrates acceptable performance under varying loads for smaller values of  $n$ , confirming the feasibility of TSS usage in practical CBDC deployments like Filia while addressing key security challenges.

**Future Work** As outlined, we identified several parts for optimization, which aim to improve the computation and communication bottlenecks. Similarly, even though the dishonest majority setting fits the CBDC setting, investigating further TSS tailored to the honest majority setting may offer reduced complexity and improved efficiency for KMN. Last, since CBDC solutions will be used for the long term, investigating post-quantum secure TSSs is essential for their future-proofness. Further efforts can focus on optimizing the computational and



communication overhead of threshold operations, particularly addressing the identified bottlenecks.

**Acknowledgments** We thank Franziska Kreitmair for early discussions on this subject and the anonymous reviewers for suggesting improvements to this paper. This work has been partially supported by the German Federal Ministry of Research, Technology and Space (BMFTR), Verbundprojekt CONTAIN (13N16582) and POST (16KIS2159); the Bavarian Ministry of Economic Affairs, Regional Development, and Energy under project 6G Future Lab Bavaria; and Horizon Europe under project SLICES-PP (10107977).

## References

- [1] [Zengo X]. *Threshold cryptography for blockchains*. URL: <https://github.com/ZenGo-X>.
- [2] 0xCarbon. *Rust-based library for Threshold ECDSA in Three Rounds*. URL: <https://github.com/0xCarbon/DKLS23>.
- [3] Damiano Abram et al. *Low-Bandwidth Threshold ECDSA via Pseudorandom Correlation Generators*. 2021. URL: <https://eprint.iacr.org/2021/1587>.
- [4] alephzero. *Threshold ECDSA*. URL: <https://gitlab.com/alephledger/threshold-ecdsa>.
- [5] Elli Androulaki et al. *A Framework for Resilient, Transparent, High-throughput, Privacy-Enabled Central Bank Digital Currencies*. 2023. URL: <https://eprint.iacr.org/2023/1717>.
- [6] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. *A Survey of ECDSA Threshold Signing*. 2020. URL: <https://eprint.iacr.org/2020/1390>.
- [7] Bank of Canada et al. *Central Bank Digital Currencies: System Design*. Nov. 2024. URL: [https://www.bis.org/publ/othp88\\_system\\_design.pdf](https://www.bis.org/publ/othp88_system_design.pdf).
- [8] Blockdaemon. *Go version of the TSM SDKv2*. URL: <https://gitlab.com/Blockdaemon/go-tsm-sdkv2>.
- [9] BNB Chain. *Threshold Signature Scheme, for ECDSA and EDDSA*. URL: <https://github.com/bnb-chain/tss-lib>.
- [10] Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. *Thresholdizing HashEdDSA: MPC to the Rescue*. 2020. URL: <https://eprint.iacr.org/2020/214>.
- [11] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. 2000. URL: <https://eprint.iacr.org/2000/067>.
- [12] Ran Canetti, Shai Halevi, and Amir Herzberg. *Maintaining Authenticated Communication in the Presence of Break-ins*. 1998. URL: <https://eprint.iacr.org/1998/012>.
- [13] Ran Canetti, Nikolaos Makriyannis, and Udi Peled. *UC Non-Interactive, Proactive, Threshold ECDSA*. 2020. URL: <https://eprint.iacr.org/2020/492>.

- [14] 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*. CCS ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1769–1787. DOI: [10.1145/3372297.3423367](https://doi.org/10.1145/3372297.3423367).
- [15] Circle Internet Financial, LLC. *Robust Threshold Signatures library*. URL: <https://github.com/circlefin/robust-tss-lib>.
- [16] Ran Cohen and Yehuda Lindell. *Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation*. 2014. URL: <https://eprint.iacr.org/2014/668>.
- [17] Consultative Group on Risk Management. *Central Bank Digital Currency (CBDC) Information Security and Operational Risks to Central Banks*. Nov. 2023. URL: <https://www.bis.org/publ/othp81.pdf>.
- [18] Ronald Cramer, Ivan Damgård, and Yuval Ishai. “Share conversion, pseudorandom secret-sharing and applications to secure computation”. In: *Proceedings of the Second International Conference on Theory of Cryptography*. TCC’05. Cambridge, MA: Springer-Verlag, 2005, pp. 342–362. DOI: [10.1007/978-3-540-30576-7\\_19](https://doi.org/10.1007/978-3-540-30576-7_19).
- [19] Ivan Damgård et al. *Fast Threshold ECDSA with Honest Majority*. 2020. URL: <https://eprint.iacr.org/2020/501>.
- [20] Jack Doerner et al. “Secure Two-party Threshold ECDSA from ECDSA Assumptions”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 980–997. DOI: [10.1109/SP.2018.00036](https://doi.org/10.1109/SP.2018.00036).
- [21] Jack Doerner et al. “Threshold ECDSA from ECDSA Assumptions: The Multiparty Case”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 1051–1066. DOI: [10.1109/SP.2019.00024](https://doi.org/10.1109/SP.2019.00024).
- [22] Jack Doerner et al. *Threshold ECDSA in Three Rounds*. 2023. URL: <https://eprint.iacr.org/2023/765>.
- [23] Yimika Erinle et al. *SoK: Design, Vulnerabilities, and Security Measures of Cryptocurrency Wallets*. 2025. DOI: [10.48550/arXiv.2307.12874](https://doi.org/10.48550/arXiv.2307.12874).
- [24] Daniel Escudero. *An Introduction to Secret-Sharing-Based Secure Multiparty Computation*. 2022. URL: <https://eprint.iacr.org/2022/062>.
- [25] Qi Feng et al. *Stateless Deterministic Multi-Party EdDSA Signatures with Low Communication*. 2024. URL: <https://eprint.iacr.org/2024/358>.
- [26] Fireblocks. *Fireblocks*. 2024. URL: <https://www.fireblocks.com/> (visited on 11/12/2024).
- [27] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup*. 2019. URL: <https://eprint.iacr.org/2019/114>.
- [28] Rosario Gennaro and Steven Goldfeder. *One Round Threshold ECDSA with Identifiable Abort*. 2020. URL: <https://eprint.iacr.org/2020/540>.
- [29] Rosario Gennaro et al. “Robust Threshold DSS Signatures”. In: *Advances in Cryptology - EUROCRYPT ’96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May*

- 12-16, 1996, *Proceeding*. Vol. 1070. Lecture Notes in Computer Science. Springer, 1996, pp. 354–371. DOI: [10.1007/3-540-68339-9\\_31](https://doi.org/10.1007/3-540-68339-9_31).
- [30] Giesecke+Devrient. *Central Bank Digital Currency G+D Filia Whitepaper*. 2024. URL: <https://pages.gi-de.com/whitepaper-filia>.
  - [31] Kilian Glas. “Evaluation of Distributed Key Generation Approaches for Threshold ECDSA Signature Systems”. Master’s Thesis. Technical University of Munich, May 2022.
  - [32] S Goldwasser, S Micali, and C Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC ’85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 291–304. DOI: [10.1145/22145.22178](https://doi.org/10.1145/22145.22178).
  - [33] Iftach Haitner et al. *Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody*. 2018. URL: <https://eprint.iacr.org/2018/987>.
  - [34] Jonathan Katz and Antoine Urban. *Honest-Majority Threshold ECDSA with Batch Generation of Key-Independent Presignatures*. 2024. URL: <https://eprint.iacr.org/2024/2011>.
  - [35] Marcel Keller. “MP-SPDZ: A Versatile Framework for Multi-Party Computation”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. DOI: [10.1145/3372297.3417872](https://doi.org/10.1145/3372297.3417872).
  - [36] Chelsea Komlo and Ian Goldberg. *Arctic: Lightweight and Stateless Threshold Schnorr Signatures*. 2024. URL: <https://eprint.iacr.org/2024/466>.
  - [37] Chelsea Komlo and Ian Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*. 2020. URL: <https://eprint.iacr.org/2020/852>.
  - [38] Yashvanth Kondi and Divya Ravi. *Separating Broadcast from Cheater Identification*. 2025. URL: <https://eprint.iacr.org/2025/052>.
  - [39] libp2p. *libp2p: A modular network stack for peer-to-peer protocols in Rust*. 2023. URL: <https://libp2p.io> (visited on 09/14/2023).
  - [40] Yehuda Lindell. *Simple Three-Round Multiparty Schnorr Signing with Full Simulatability*. 2022. URL: <https://eprint.iacr.org/2022/374>.
  - [41] Lockness. *State-of-art threshold ECDSA in Rust*. URL: <https://github.com/LFDT-Lockness/cggmp21> (visited on 09/08/2025).
  - [42] Lúcas Meier. *Threshold ECDSA via Triples*. URL: <https://github.com/cronokirby/cait-sith>.
  - [43] MPC Alliance. *MPC Alliance*. 2024. URL: <https://www.mpcalliance.org/> (visited on 09/08/2025).
  - [44] Nakasendo Project. *Nakasendo*. URL: <https://web.archive.org/web/20200424131900/https://nakasendoproject.org/docs/HighLevelFunctions/index.html>.
  - [45] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT’99. Prague, Czech Republic: Springer-Verlag, 1999, pp. 223–238.

- [46] Michaella Pettit. *Efficient Threshold-Optimal ECDSA*. 2021. URL: <https://eprint.iacr.org/2021/1386>.
- [47] Jason Resch et al. *A Platform for Robust Threshold Cryptography*. URL: <https://github.com/jasonkresch/protect>.
- [48] Filip Rezabek et al. “EnGINE: Flexible Research Infrastructure for Reliable and Scalable Time Sensitive Networks”. In: *Journal of Network and Systems Management* 30.4 (Sept. 2022), p. 74. DOI: [10.1007/s10922-022-09686-0](https://doi.org/10.1007/s10922-022-09686-0).
- [49] Filip Rezabek et al. *Multilayer Environment and Toolchain for Holistic NetwOrk Design and Analysis*. Nov. 2023. DOI: [10.48550/arXiv.2310.16190](https://doi.org/10.48550/arXiv.2310.16190).
- [50] Tim Ruffing et al. “ROAST: Robust Asynchronous Schnorr Threshold Signatures”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 2551–2564. DOI: [10.1145/3548606.3560583](https://doi.org/10.1145/3548606.3560583). URL: <https://doi.org/10.1145/3548606.3560583>.
- [51] Silence Laboratories. *Multi-Party Threshold Signature Library for ECDSA*. URL: <https://github.com/silence-laboratories/silent-shard-dkls23-11> (visited on 09/08/2025).
- [52] Safeheron. *C++ implementation of (t,n)-Threshold Signature Scheme*. URL: <https://github.com/Safeheron/multi-party-sig-cpp>.
- [53] Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. *PEReDi: Privacy-Enhanced, Regulated and Distributed Central Bank Digital Currencies*. 2022. URL: <https://eprint.iacr.org/2022/974>.
- [54] Kiarash Sedghighadikolaie and Attila Altay Yavuz. *A Comprehensive Survey of Threshold Signatures: NIST Standards, Post-Quantum Cryptography, Exotic Techniques, and Real-World Applications*. 2024. DOI: [10.48550/arXiv.2311.05514](https://doi.org/10.48550/arXiv.2311.05514).
- [55] Adi Shamir. “How to share a secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: [10.1145/359168.359176](https://doi.org/10.1145/359168.359176). URL: <https://doi.org/10.1145/359168.359176>.
- [56] Yang Shi et al. “Threshold EdDSA Signature for Blockchain-based Decentralized Finance Applications”. In: *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*. RAID ’22. Limassol, Cyprus: Association for Computing Machinery, 2022, pp. 129–142. DOI: [10.1145/3545948.3545977](https://doi.org/10.1145/3545948.3545977). URL: <https://doi.org/10.1145/3545948.3545977>.
- [57] Douglas R. Stinson and Reto Stroh. “Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates”. In: *Information Security and Privacy*. Ed. by Vijay Varadharajan and Yi Mu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 417–434.
- [58] Taurus SA. *Implementation of protocols for threshold signatures*. URL: <https://github.com/taurushq-io/multi-party-sig>.

- [59] Dmitry Varlakov and Jonathan Katz. *CGGMP21 In Rust, At Last*. Mar. 2024. URL: <https://www.dfns.co/article/cggmp21-in-rust-at-last> (visited on 09/08/2025).