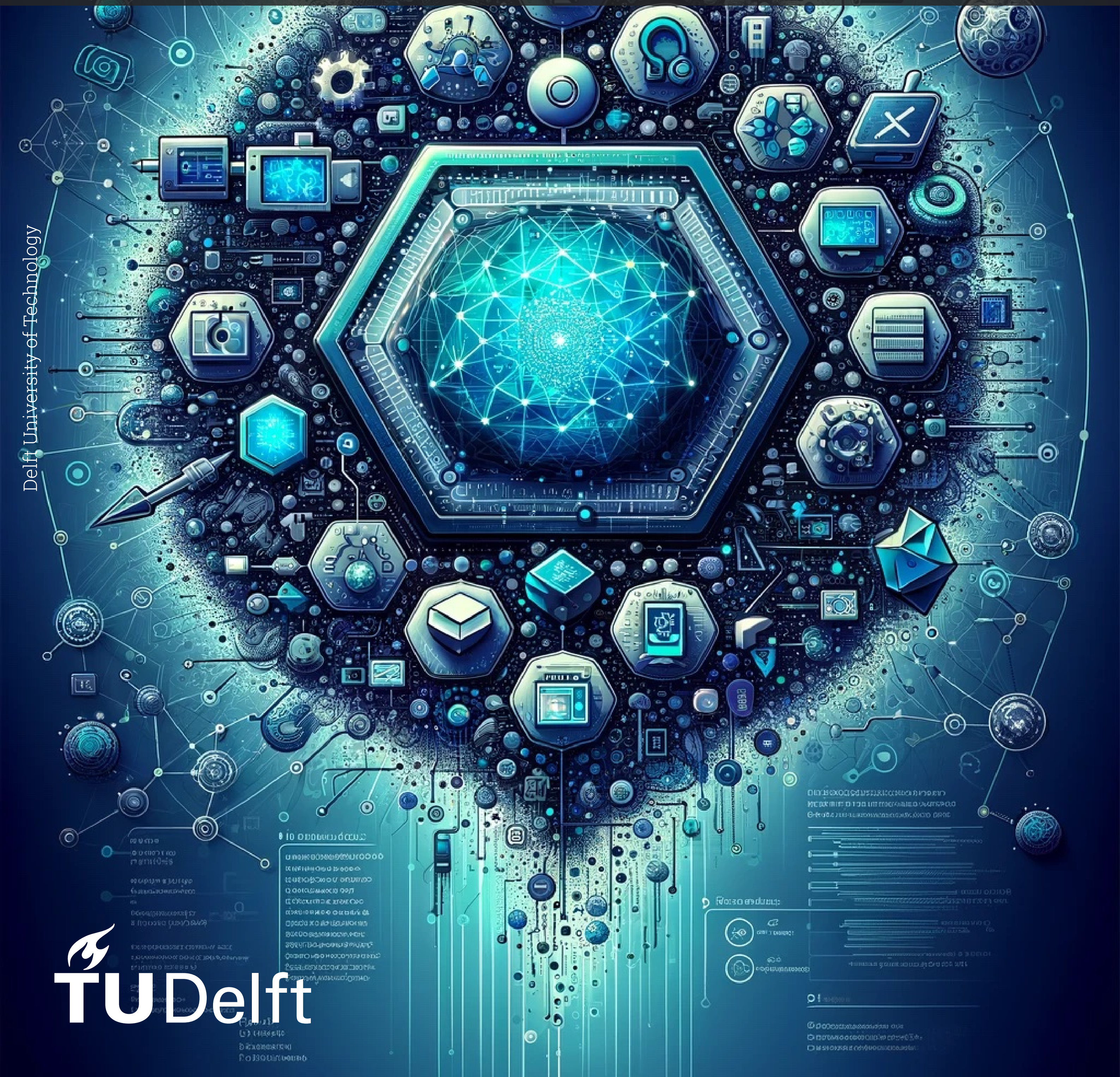
Completely FROST-ed

IoT issued FROST signature for Hyperledger Fabric blockchain

Mostafa Khattat

Delft University of Technology



TUDelft

Completely FROST-ed

IoT issued FROST signature for Hyperledger Fabric blockchain

by

Mostafa Khattat

TU Delft

Acknowledgement

I would like to express my sincere gratitude to all who contributed to the successful completion of my master's thesis.

Special thanks are extended to Dr. Roland Kromes for his exceptional mentorship and invaluable support. His expertise helped shape the foundation of this thesis and his insightful feedback and constructive criticism helped me to navigate through various obstacles encountered during the research process. I am sincerely grateful for his understanding and flexibility during unforeseen delays, which allowed me to successfully complete my master's thesis.

My thanks also extend to my thesis advisor Prof.dr Georgios Smaragdakis, and my daily supervisor Dr. Kaitai Liang for their insightful feedback and constructive contributions.

Finally, I would like to express my heartfelt gratitude to my family and friends. Their unconditional love, patience, and continuous support have been a constant source of strength throughout this endeavor.

Mostafa Khattat
Delft, April 2024

Contents

Abstract

Threshold signatures play a crucial role in the security of blockchain applications. An efficient threshold signature can be applied to enhance the security of wallets and transactions by enforcing multi-device-based authentication, as this requires adversaries to compromise more devices to recover the key. Additionally, threshold signatures can protect user privacy, for instance, by enabling anonymous transaction signing on behalf of a group of users sharing a blockchain wallet. This study conducts a comprehensive analysis of threshold signature schemes, identifying FROST as the premier choice due to its performance efficiency, improved energy consumption, and practical feasibility on mid-range IoT devices and smartphones as demonstrated through empirical testing. Furthermore, we introduce a protocol for integrating FROST with Hyperledger Fabric v3.0, aimed at enhancing IoT devices' ability to interact with blockchain networks through efficient transaction signing. Our experiments reveal that an IoT network of five devices, under optimal network conditions, can sign a transaction and commit it to the Hyperledger Fabric network within 3.2 seconds, leveraging a 2-second batch timeout configuration.

1 Introduction

In the evolving landscape of digital transformation, the integration of blockchain technology in various sectors has become a key point for both industry and academia. With over 40% of organizations worldwide adopting blockchain for digital currencies, asset tracking, and digital identification, the need for efficient and secure transaction signing mechanisms has never been more critical [13].

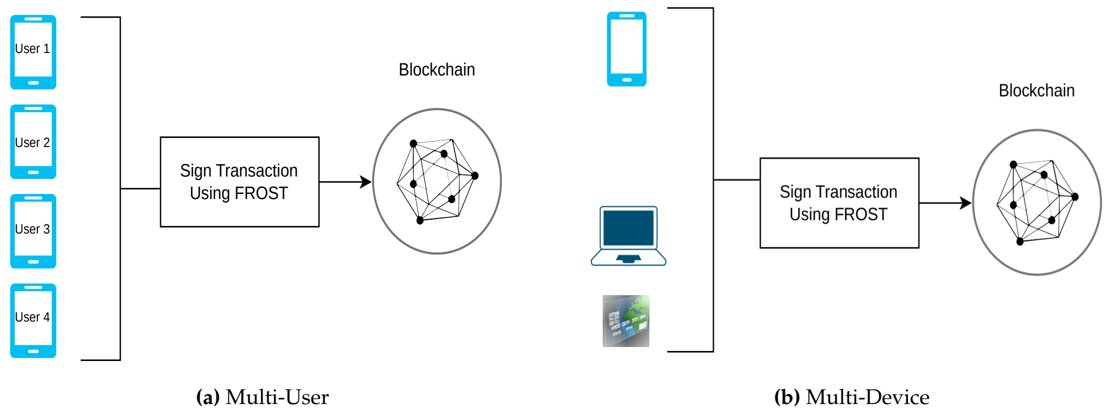
The conventional approach to managing digital signatures and keys in blockchain applications involves storing the cryptographic keys on a single device. This method, while straightforward, introduces a single point of failure; that is, if the device is compromised, the keys - and consequently, the digital assets or permissions they protect - are at risk [3].

One alternative solution to mitigate the risk of a single point of failure is to allow a group of signers to produce a joint signature on a common message. The most straightforward and trivial approach to creating a multi-signature involves each signer generating an individual signature with their private key and then aggregating these signatures, this method results in a multi-signature size that increases linearly with the number of signers. This allows for the verification of the signature's validity using the message and the collective public keys of all participating signers [31].

While the traditional multi-signatures are functional, they are insufficient in terms of scalability and efficiency, leading to increased verification time and storage requirements on the blockchain. Additionally, this approach can potentially lead to privacy issues, as it reveals the identities of the signers.

This will set the stage for our investigation into threshold signature schemes, particularly the FROST algorithm, to find a suitable alternative, especially in the context of the Internet of Things (IoT) devices. Threshold signatures, or "t-of-n" signatures, offer a more streamlined approach where a single signature, verifiable by a common public key, suffices for transaction validation. This does not only reduces the blockchain's storage but also increases security by distributing the signing capability among multiple participants or devices.

In this work, we provide a comprehensive survey of recent threshold signatures, focusing on two approaches: ECDSA- and Schnorr-based schemes. We show the performance, characteristics differences and a summary of key and signature generation. Additionally, it is important for the algorithms running on IoT devices to be energy efficient. We evaluate how FROST performs in terms of energy consumption compared to two other popular threshold signature schemes, namely GG20 [27] and DKLS [20].



(a) Multi-User **(b)** Multi-Device

Figure 1.1: Frost threshold signature use cases in blockchain context

1.1. Problem Description and Motivation

The integration of blockchain technology in IoT applications presents unique challenges, notably in terms of resource constraints and efficiency. Our work is motivated by the need to address the limitations of ECDSA-based threshold signatures, which are characterized by their high execution time, high energy consumption, and operational complexity. With potentially thousands of devices needing to authenticate transactions or communications securely, the scalability of the signing mechanism becomes crucial. The traditional approaches, especially those involving multi-signatures, lack the scalability.

By focusing on FROST, a Schnorr signature-based scheme, we aim to explore its adaptability and performance benefits, particularly in IoT environments. Despite the predominant use of ECDSA in current blockchain technologies, the promising characteristics of FROST, coupled with the architectural flexibility of permissioned blockchains such as Hyperledger Fabric, present a compelling case for our research.

Furthermore, this work is inspired by applying two real-life use case scenarios in the context of an IoT-Blockchain system in which integrating FROST threshold signature can make it feasible. Figure 1.1 shows an overview of the two use cases. Below we explain each in more detail.

Multi-Device Scenario

In the first scenario, the Multi-device wallet, we focus on enhancing the security of a blockchain wallet by distributing its secret key across multiple devices owned by a single user. The Multi-Device scenario illustrates a form of distributed multi-factor authentication (MFA) within the IoT blockchain applications. These devices can be a smartwatch, smartphone or laptop. For example, by using a 2-out-of-5 threshold signature scheme, the protocol requires collaboration among at least two out of five devices to authorize a transaction with the blockchain wallet's secret key. This approach ensures the security of the wallet against unauthorized access. The distributed nature of the secret key across devices introduces a layer of protection that is resilient to the compromise of any single device, ensuring that the blockchain wallet's integrity is maintained even in the event of partial security breaches [54].

Multi-User Scenario

The second scenario, the Multi-user wallet, focuses on enhancing privacy protection and enabling shared ownership of a blockchain wallet among multiple users. In this configuration, the secret key of a blockchain wallet is divided among the participants. For instance, utilizing a 3-out-of-5 threshold signature scheme requires the cooperation of at least three out of five users to execute a transaction, thus enabling joint control over the shared wallet. This model not only strengthens the privacy of users but also introduces an anonymous signing process. Within this framework, users collaboratively produce a single signature for the transaction. This collective signature is indistinguishable from those generated by individual users. Importantly, the blockchain, which performs the usual verification process, is unaware that the signature is the product of a collaborative effort among multiple users. The key to this anonymity lies in the off-chain generation of the threshold signature. Unlike traditional multi-signatures, which are processed on-chain and can reveal the involvement of multiple signers, the off-chain approach of threshold signatures conceals the fact that the signature was generated by a group. As a result, the generated signature, verifiable by the blockchain wallet's public key, hides the identities and the exact number of users involved in the transaction, thus providing anonymity and preventing any external party from deducing the internal dynamics of the wallet's ownership [54].

1.2. Contributions

The main contributions to this work are as follows:

- A comprehensive survey of recently proposed threshold schemes, ECDSA and Schnorr, in IoT context.
- We provide APIs deployed on top of Hyperledger Fabric SDK, enabling transaction signing with the FROST signature algorithm and the registration of a group of devices to the Hyperledger Fabric network.
- Evaluating energy consumption of three threshold signature schemes, GG20, DKLS, and FROST, in IoT devices. We provide insights into their operational efficiencies. This aspect of our research not only highlights FROST's suitability for IoT devices but also emphasizes its potential to revolutionize transaction signing in blockchain networks.
- Deployment of an IoT-enabled Android-based application combining FROST, and Hyperledger Fabric SDK.
- Testing FROST signature verification on Hyperledger Fabric v3.0.

1.3. Paper Structure

The rest of the paper is organized as follows: Chapter 2 describes the preliminaries of digital signatures, FROST signature, blockchain technologies overview and an overview of Hyperledger Fabric and its components. Chapter 3 presents Blockchain and IoT. Furthermore, chapter 4 provides a state-of-the-art survey of threshold signatures. Moreover, our proposed protocol is presented in chapter 5. The experiments and results of the proposed protocol are provided in chapter 5. Finally, chapter 7 discusses and chapter 8 concludes the paper.

2 Preliminaries

In this section, we describe the preliminaries of digital signatures, the FROST signature, and blockchain technologies and highlight the Hyperledger Fabric blockchain.

2.1. Digital Signatures

Digital signatures play an important role in ensuring the authenticity and integrity of electronic documents and transactions. Essentially, a digital signature is a cryptographic mechanism that verifies the origin and integrity of digital messages or documents. Similar to handwritten signatures, digital signatures provide a means for individuals or entities to sign electronic documents [66].

A digital signature involves the use of asymmetric cryptography, where a pair of keys, namely a private key and a public key, are utilized. The private key is known only to the signer and is used to generate the signature, while the corresponding public key is shared openly and used to verify the signature.

Signing

To sign a document, the signer uses their private key to generate a unique cryptographic hash of the document. This hash, along with other relevant information, is encrypted using the signer's private key to produce the digital signature.

Verification

To verify the signature, the recipient of the document uses the signer's public key to decrypt and obtain the hash value. Subsequently, the recipient independently computes the hash of the received document. If the computed hash matches the decrypted hash obtained from the signature, the signature is considered valid, confirming the document's authenticity and integrity.

In this section, we explore two digital signature schemes: the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Schnorr Signature.

2.1.1. ECDSA Signature

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve variant of the Digital Signature Algorithm (DSA). Below, we briefly explain the procedure for generating and verifying an ECDSA signature [32]:

Suppose Alice wants to sign a message m and send the signature to Bob. They both agree on the elliptic curve parameters (G, n) where G is a base point on the curve and n is the multiplicative order of G and is prime. Furthermore, Alice generates a pair of public key Q_A and a private key d_A where $Q_A = d_A \times G$. Finally, to verify if Alice's signature is valid, Bob must have a copy of her public key Q_A.

Signature Generation

1. Select a cryptographic secure pseudorandom integer k where $1 \leq k \leq n-1$
2. Compute the curve point $(x_1, y_1) = k \times G$.
3. Compute $r = x_1 \bmod n$. If $r = 0$ then start over.
4. Compute $k^{-1} \bmod n$.
5. Compute $e = \mathrm{HASH}(m)$ where m is the message and HASH is a cryptographic hash function e.g., SHA-3 [49].
6. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then start over.
7. The signature for the message m is the pair (r, s).

Signature Verification

1. Verify that r and s are integers where $1 \leq r, s \leq n-1$. Otherwise, reject the signature.
2. Compute $e = HASH(m)$ where HASH is the same cryptographic hash function used in the signature generation.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = u_1 G + u_2 Q$. If $X = O$, then reject the signature where O is the identity element.
6. Compute $v = x_1 \bmod n$ where $X = (x_1, y_1)$.
7. if and only if $v = r$ then accept the signature.

2.1.2. Schnorr Signature

The Schnorr Signature is one of the earliest discrete logarithmic-based signature schemes derived from the Schnorr identification scheme [61, 60]. Pointcheval and Stern proved the security under the assumption of Discrete Logarithm [51, 50] in the Random Oracle Model [12]. Originally the Schnorr signature was covered by patent US4995082A [59] . However, the patent expired in February 2010.

Similar to ECDSA, we first generate a pair of public and private key

- Select the private key x from $Z_q \setminus \{0\}$.
- Compute the public verification key $y = g^x$.

Signature Generation
To sign a message m:

1. Chose a random k from Z_q.
2. Compute $r = g^a$.
3. Compute $e = HASH(m||r)$ where || denotes concatenation or padding.
4. Compute $s = k + ex \bmod q$.
5. The signature for the message m is the pair (s, e).

Signature Verification
Given a message m and a signature (s, e):

1. Compute $r_v = g^s y^e$.
2. Compute $e_v = HASH(r_v||m)$.
3. if and only if $e_v = e$ then accept the signature.

2.2. FROST: Flexible Round-Optimized Schnorr Threshold Signatures

FROST is a threshold signature scheme built upon the Schnorr signature algorithm. FROST requires only two rounds of communication for both key generation and the signing protocol. Additionally, it offers the potential for optimization to a single round by incorporating a preprocessing step that can be performed before the message is known [35]. The core characteristic of FROST is its trade-off between network efficiency and robustness. However, not all participants are honest. There are cases where some misbehaving participants do not follow the protocol or contribute malformed values during the protocol. While FROST is capable of identifying misbehaving participants (primarily during the signing phase), it comes at the cost of potentially having to re-run the signing protocol when misbehavior is detected [35]. Below, we explain briefly the FROST key generation and signing algorithm:

A semi-trusted signature aggregator (SA) can be used to reduce the communication overhead. The role of (SA) doesn't give more privileges than the adversary and can be performed by either any of the participants or a third party as long as they know the participants' public key shares. The SA has the responsibility to report the misbehaving participants and to publish the group's signature at the end of the protocol.

However, in some scenarios where having a central SA is not feasible, FROST can be used without a signature aggregator. Participants need to simply perform a broadcast instead of using a SA for coordination.

The FROST protocol consists of two sub-protocols:

- **Key generation** in which all participants collaboratively generate a group's private key with a corresponding public key. Note that the group's private key is never constructed. Instead, each participant receives a long-lived private share which will be used to compute a verification share.
- **Signing** in which any subset of participants equal to or greater than the threshold can sign a message.

Below, we describe these two sub-protocols in more detail.

Key Generation

FROST builds upon Pedersen's Distributed Key Generation (DKG) protocol [48], where each participant concurrently executes Feldman's Verifiable Secret Sharing (VSS) [24]. FROST requires participants to prove knowledge of their secret shares through zero-knowledge proofs [28], specifically instantiated as Schnorr signatures to protect against potential rogue-key attacks in a setting where the threshold setting $t \geq n/2$ [11].

In the first round of key generation, after participants collectively agree on the threshold value t and the total number of participants n, each participant P_i chooses a secret value a_{i0} along with $t - 1$ random coefficients. These coefficients are utilized to generate a polynomial of degree $t - 1$.

Each participant P_i computes a zero-knowledge proof (ZKP) for their a_{i0} and constructs a vector of public commitments for all coefficients. Participants broadcast their ZKPs and commitment vectors, and each participant verifies the received ZKPs using the first component of the public commitment. If verification fails, the protocol is aborted.

In the second round of key generation, each participant securely sends to every other participant a secret share which is the shares of their secret a_{i0}. The shares are validated by each participant using the public commitment vector. The protocol aborts if the validation check fails.

Finally, participants compute their long-lived private signing shares and their public verification shares Y_i. The group public key Y is derived from the first component of the public commitment vector contributed by each participant.

Signing Protocol

The signing protocol for FROST is divided into a pre-processing phase and a single-round signing phase (or combined into a two-round protocol). FROST prevents known forgery attacks, notably addressing the threat posed by Drijvers et al. [23]. The binding technique ensures that each participant's response is linked to a specific message, set of participants, and commitments used in a signing operation, preventing malicious actions without restricting concurrency.

Pre-processing Phase: This phase can be executed before the message is known and is designed to support multiple signing operations (π). During this phase, each participant calculates π private single-use nonces (d_{ij}, e_{ij}) along with their corresponding public commitments (D_{ij}, E_{ij}), which are then broadcasted to all other participants.

Signing Phase: In this phase, a message m is known, and at least t signers are selected to sign the message. Each signer selects the first available commitment for every other participant with their corresponding identifier i in the format $B = (D_i, E_i, \mathrm{id}_i)$ and constructs an ordered list of tuples $<B, m>$. Subsequently, signers calculate a set of binding values $\rho_i = H_1(i, m, B_i)$ using a hash function H_1.

- The group commitment R is computed alongside the challenge $c = H_2(R, Y, m)$, utilizing another hash function H_2.
- Each participant calculates their response $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$, where s_i represents the long-lived secret share and λ_i is the Lagrange coefficient for id_i within the set of signers.
- Participants transmit their respective z_i values to all other participants.

Aggregation and Verification: Following reception of z_i values, participants verify the validity of each z_i using their public verification share Y_i. Once verified, the group's response is computed as $z = \sum_i z_i$.
Group Signature: The group's signature on message m is represented as (R, z), which can be verified using the standard Schnorr verification algorithm alongside the group's public key Y.

2.3. Blockchain

Blockchain is a decentralized and distributed ledger consisting of a sequence of blocks to keep track of all transaction records. These blocks are chained together using a cryptographic hash of a previously created block which guarantees causality. This means if someone wants to submit the same transaction twice, they have to change all the blocks before the latest block since the first block is called the Genesis block and has no parent.

2.3.1. Types of Blockchain

A blockchain network can be divided into three main categories based on their permission: public, private and consortium [70, 19]. Public blockchains are open to everyone, and allow any user to participate in the core activities of the network such as validating transactions, reading and consensus process resulting in immutable data storage [68]. In this type of blockchain, the entire node must agree on the appended blocks which means it takes more time to process and record the block to the blockchain [40]. Bitcoin [43] and Ethereum [25] are examples of such public blockchain networks.

A private blockchain, on the other hand, is controlled by a centralized network and only some nodes are allowed to join the consensus process. The private blockchain can have a higher throughput compared to the public blockchains since it the validation happens by fewer nodes with a high processing power as opposed to a large number of nodes in public blockchain networks which could be running on less powerful systems [74]. In this type of network, the consortium or the company running the blockchain can easily modify the network such as changing the rules, reverting the transactions, etc. Furthermore, since the validators are known the risk of a 51% attack [8] does not apply. Hyperledger Fabric is an example of such a private blockchain network [6].

Consortium blockchains represent an interesting architectural model that combines the transaction efficiency and privacy features of private blockchains with the decentralized governance characteristic of public blockchains. The key distinction between private and consortium blockchains lies in their governance structure and infrastructure. In private blockchains, a single entity has full control over the entire system, while consortium blockchains distribute authority among multiple members. Consequently, the infrastructure of private blockchains tends to be centralized, resembling traditional distributed databases where data is replicated across nodes owned by a single entity. On the other hand, consortium blockchains are deployed in a decentralized fashion across multiple hardware nodes managed by different owners or companies. Furthermore, data within consortium blockchains is not necessarily homogeneous among consortium nodes, as certain transactions can be kept private and shared only among specific subsets of participants, resulting in knowledge fragmentation across the network. This nuanced approach to governance and data sharing makes consortium blockchains well-suited for collaborative endeavors involving multiple stakeholders with varying levels of trust and authority [19].

2.3.2. Consensus Algorithms

Reaching an agreement within a blockchain network, especially where nodes are inherently untrustworthy and the environment is decentralized, is a challenging Byzantine Generals (BG) problem [36]. The BG problem illustrates the difficulty of achieving unanimous agreement in a distributed system where some participants may act maliciously or fail to communicate. In the context of blockchain, this issue is compounded by the fact that once transactions are added to a block and the block is verified by the network, it becomes immutable or in the case of public blockchains deleting these records is not feasible.

The network is designed to function reliably even amidst adversarial users in a trustless environment.

Over time, a variety of consensus algorithms have been developed, each designed to address specific challenges of decentralization, security, and efficiency. Although the number and complexity of these algorithms continue to grow with the evolution of blockchain technology, we focus on some of the widespread adoption options in the blockchain networks such as Proof of Word (PoW), Proof of Stake (PoS) and Practical Byzantine Fault Tolerance (PBFT).

Proof of Work (PoW)

Proof of Work (PoW), a consensus mechanism first introduced by Satoshi Nakamoto in 2009 and foundational to Bitcoin's operation, represents a strategy for achieving consensus in decentralized networks [43]. At its core, PoW requires nodes, or miners, to perform extensive computational work to solve mathematic calculations. This process, known as mining, involves calculating a hash value for the block header, which includes a unique number called a nonce. The objective is to find a nonce that results in a hash value meeting a specific condition, such as having a certain number of leading zeros. This condition ensures that the solution is difficult to achieve but easy for the network to verify.

The hash function itself is designed to be a one-way, computationally demanding task that validates the transactions within a block. When a miner successfully solves the puzzle, they broadcast the block to the network, which then verifies the solution. If accepted, the new block is added to the blockchain, and the miner is rewarded [57]. Additionally, the difficulty of the puzzles can be adjusted, offering scalability to accommodate network growth.

However, the PoW mechanism is not without its drawbacks. The intensive computational effort required for mining consumes a significant amount of energy, raising concerns about sustainability [74]. Furthermore, the reliance on specialized hardware for mining concentrates power among those who can afford such equipment, potentially compromising the decentralized ethos of blockchain. The time-consuming nature of puzzle-solving also limits transaction throughput, making PoW less suited for networks that demand high transaction volumes and speeds [5].

In response to some of the drawbacks regarding intensive computational effort, alternative PoW protocols, such as Primecoin, have been developed to ensure that the computational work performed during mining has secondary applications, such as searching for chains of prime numbers valuable for mathematical research [33].

Proof of Stake (PoS)

Proof of Stake (PoS) emerged as a consensus algorithm to address the significant energy consumption and inefficiency inherent in Proof of Work (PoW) systems. Introduced with the Peercoin cryptocurrency in 2011, PoS represents a more sustainable and scalable approach to blockchain consensus by determining the creator of the next block through mechanisms that consider the user's stake in the cryptocurrency [10].

Unlike PoW, which relies on computational power to validate transactions and create new blocks, PoS selects validators based on the amount of cryptocurrency they hold and are willing to "stake" as collateral. This process can involve various combinations of random selection, the stake's size, and its age, reducing the need for energy and computation-intensive mining activities. Consequently, validators in PoS systems are rewarded not through new cryptocurrency creation but via transaction fees [57].

However, PoS is not without its drawbacks. Centralization concerns arise since the system inherently favors those with larger stakes, potentially leading to a dominance of the network by wealthier nodes. Furthermore, the "nothing at stake" problem posits a scenario where validators have little to lose in acting maliciously, such as by supporting multiple blockchain forks to receive more transaction fees [57].

Some solutions have been proposed to overcome some of the drawbacks of PoS. For example, in Peercoin [34] protocol, block generation is based on coin age. In this approach, the probability of mining the next block linearly increases by the amount of unspent coins.

Despite some of the benefits of the PoS approach such as higher transaction throughput, reduced block creation times, independence from specialized hardware and being more energy-efficient and scalable compared to the PoW, it is not trivial to transition from PoW to PoS as can be seen by the ongoing effort of Ethereum's gradual shift from Ethash (a PoW algorithm) [25] to Casper (a PoS algorithm) [72].

Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) addresses the Byzantine General's Problem, an issue in distributed computing systems where nodes must agree on a single course of action, even in the presence of malicious actors. This consensus mechanism is particularly relevant in today's digital ecosystem, where the increasing reliance on online services amplifies the potential impact of software errors and malicious attacks [17].

PBFT operates on a state machine replication model, ensuring that all nodes in the network agree on the state of the system. It achieves consensus through a voting process where a block is added to the chain only if more than two-thirds of the nodes express a favorable opinion. This threshold allows the system to tolerate up to one-third of the nodes acting maliciously or failing. For instance, in a network with one malicious node, at least four nodes must reach an agreement for consensus to be achieved. This mechanism enables PBFT to provide a faster and more cost-effective solution to consensus than Proof of Work, without requiring any form of asset stake as in Proof of Stake [57].

Despite its advantages, including energy efficiency and high throughput, PBFT faces challenges such as potential delays due to the requirement for comprehensive voting across all nodes.

2.3.3. Smart Contracts

Smart contracts, conceptualized by Nick Szabo, execute contracts with the terms of the agreement between parties directly written into lines of code [67]. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism.

Zheng et al. divides the life cycle of smart contracts into four phases [76]:

- **Creation:** The process begins with the negotiation of contract terms among the involved parties, including obligations, rights, and prohibitions. Lawyers or counselors draft the initial agreement in natural language, which software engineers then translate into computer code. This phase is iterative, involving multiple rounds of discussion to refine the contract details, ensuring that all parties' expectations are met and accurately represented in the final smart contract [75].

- **Deployment:** Once the smart contract is ready, it is deployed on a blockchain platform. Due to the immutable nature of blockchain, any modifications post-deployment require the creation of a new contract [75].

- **Execution:** The smart contract actively monitors and evaluates the contractual conditions after deployment. When specific conditions are met, the contract automatically executes the agreed-upon functions. This phase emphasizes the autonomous execution of contracts based on predefined logic, ensuring that all actions are recorded as immutable transactions on the blockchain [75].

- **Completion:** Upon execution, the smart contract updates the state of all parties involved and unlocks the previously frozen digital assets, facilitating the transfer of assets from one party to another as per the contract terms. This final phase marks the fulfillment of the contractual obligations and the conclusion of the smart contract's lifecycle, with all relevant transactions and updated states securely stored on the blockchain [75].

2.4. Hyperledger Fabric

Hyperledger Fabric is one of the most popular permissioned blockchains designed to meet the needs of enterprise and business applications, preserving the advantageous features of blockchain technology while addressing aspects that may be less suited for enterprise environments [37].

Hyperledger Fabric serves as a distributed ledger platform and facilitates the execution of distributed applications written in general-purpose programming languages such as GO, Java, and NodeJS. Hyperledger Fabric introduces a different approach for transactions known as execute-order-validate [7].

At its core, the Fabric consists of multiple nodes that form a network. These nodes communicate with each other and execute programs, known as smart contracts or chaincode, maintain state, and manage ledger data.

The nodes in the Fabric fulfill three distinct roles: client, peers, and Ordering Service Nodes (OSNs). Clients submit transaction proposals, peers execute, and validate transactions, and maintain the state and the ledger data. Further, the OSNs collectively ensure consistency and integrity throughout the system. They provide communication services such as "broadcast" and "deliver" calls to establish order on transactions. Note that the orderer nodes are unaware of the application state and do not partake in transaction execution or validation, simplifying the consensus process and allowing for modularity in consensus protocol selection.

The system state which reflects the latest state of the blockchain is represented as a versioned key-value store (KVS), where chaincode manipulates data through "put" and "get" operations. Meanwhile, the ledger serves as a comprehensive record of all valid and invalid state changes, organized into blocks by the ordering service.

Additionally, Fabric supports the concept of channels, enabling the creation of multiple blockchains connected to the same ordering service. Each channel functions as an independent blockchain with its own set of members, allowing for state partitioning [7].

2.4.1. The Membership Service Provider

Since Fabric is a permissioned network, participants in the blockchain network need to prove their identity. The Membership Service Provider (MSP) manages identities and roles within the network, ensuring that participants can prove their identity and authorization to interact with the blockchain. MSP uses a public key infrastructure (PKI) to establish a chain of trust, allowing Certificate Authorities to issue identities in the form of digital certificates. Each identity is associated with specific roles such as admin, peer, client, or orderer. These identities are recognized by the network and used to endorse transactions and validate the integrity of network actions. Moreover, the MSPs permit the recognition of revoked identities, enhancing security measures by preventing unauthorized access or actions within the network.

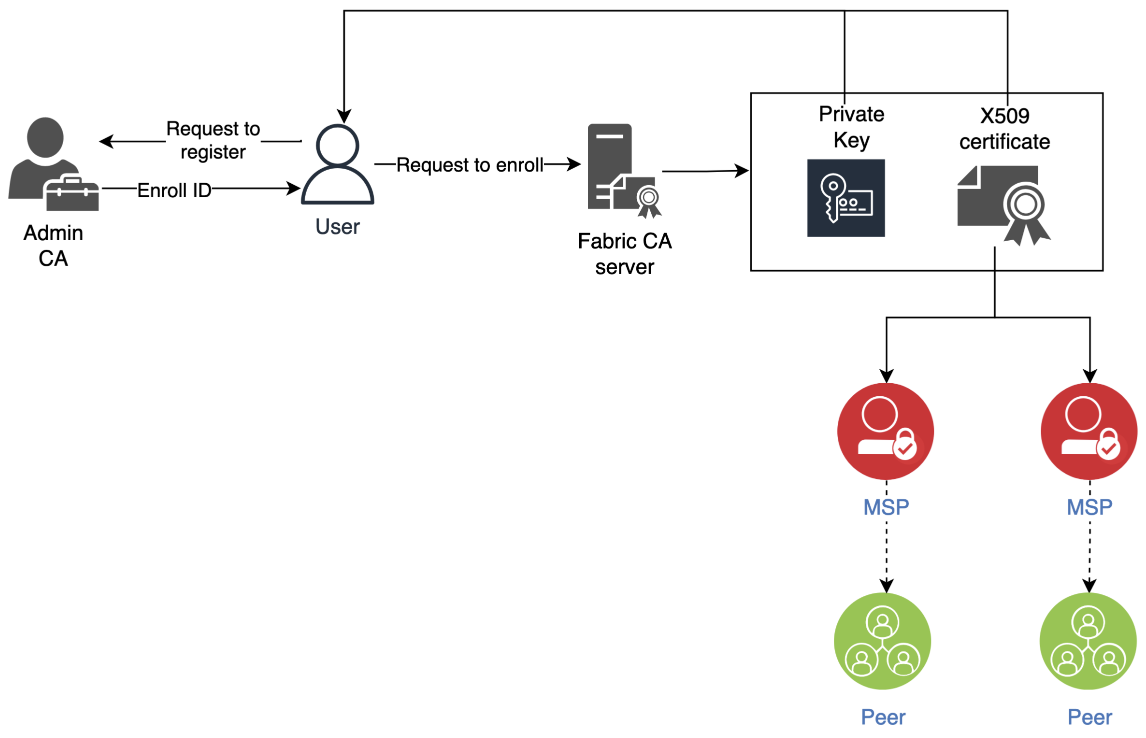


Figure 2.1: Current process of registering and enrolling a user in Fabric

2.4.2. Overview of registration and enrollment

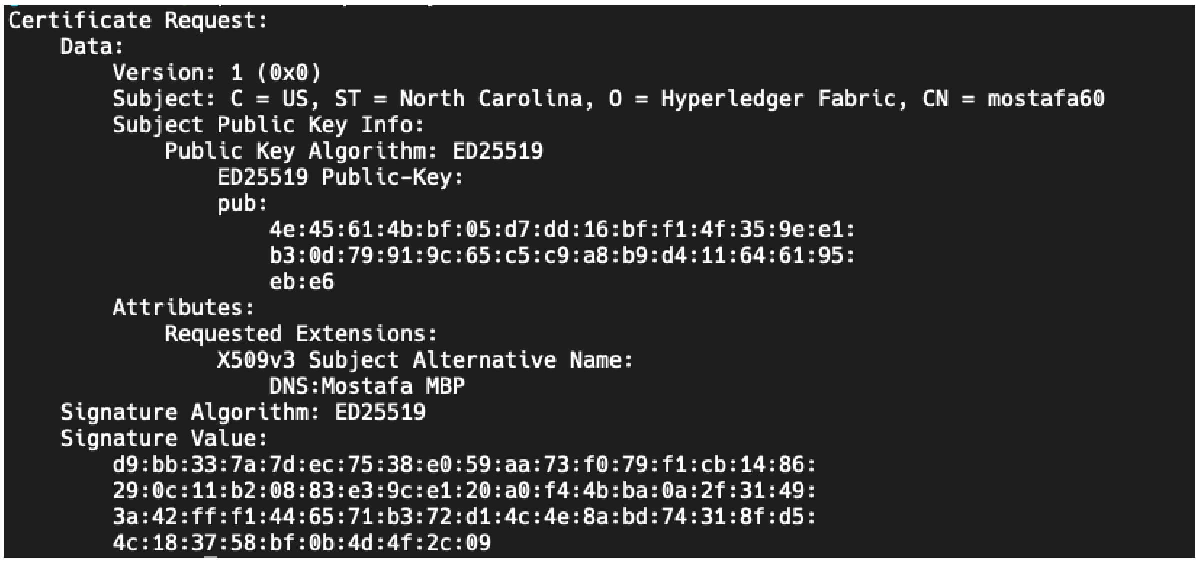
In Hyperledger Fabric, the Certificate Authority (CA) is responsible for dispensing X.509 certificates, which serve as digital identities for network participants. The Fabric CA provides two essential features: registration of identities, and issuance of enrollment certificates.

During registration, the CA administrator assigns an enroll ID and secret (similar to a username and password) to an identity, along with specifying its role, affiliation, country, and any required attributes. Once the identity is registered, the Fabric CA generates a pair of public/private keys for the user. Together with the parameters provided by the CA administrator, the user generates a certificate Signing Request (CSR). The CSR contains three main components:

- **Certificate Request Information:** This section contains the enroll ID, public key, host, serial number, and other relevant user information.
- **Signature Algorithm:** The signature algorithm identifier e.g, ECDSA, ED25519, etc.
- **Digital Signature:** The signature on the certificate requests information that is signed by the user's private key.

Figure 2.2 shows an example of a CSR. Finally, the CSR will be processed by the Fabric CA to generate a public X509 certificate and will be sent back to the user and the MSP. The private key, however, is kept by the user and is used to sign a transaction. The other nodes on the network can then verify the transaction proposal by using the public X509 certificate stored in MSP. Figure 2.1 shows an overview of this process.

Admin identities within the network are also subject to registration and enrollment processes. Admin identities are registered and enrolled with an "organization CA," which generates identity certificates for both admins and nodes. Additionally, nodes within the network are required to be registered and



```
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = US, ST = North Carolina, O = Hyperledger Fabric, CN = mostafa60
        Subject Public Key Info:
            Public Key Algorithm: ED25519
                ED25519 Public-Key:
                pub:
                    4e:45:61:4b:bf:05:d7:dd:16:bf:f1:4f:35:9e:e1:
                    b3:0d:79:91:9c:65:c5:c9:a8:b9:d4:11:64:61:95:
                    eb:e6
        Attributes:
            Requested Extensions:
                X509v3 Subject Alternative Name:
                    DNS:Mostafa MBP
    Signature Algorithm: ED25519
    Signature Value:
        d9:bb:33:7a:7d:ec:75:38:e0:59:aa:73:f0:79:f1:cb:14:86:
        29:0c:11:b2:08:83:e3:9c:e1:20:a0:f4:4b:ba:0a:2f:31:49:
        3a:42:ff:f1:44:65:71:b3:72:d1:4c:4e:8a:bd:74:31:8f:d5:
        4c:18:37:58:bf:0b:4d:4f:2c:09
```

Figure 2.2: An example of a CSR

enrolled with a TLS CA to create a public/private TLS key pair for securing communications. This TLS key pair is utilized by nodes for signing and encrypting communications.

3 Blockchain And IoT

The Internet of Things (IoT) is an expansive network that extends internet connectivity beyond traditional devices like computers and smartphones to a vast array of objects, devices, sensors, and everyday items. This concept, first introduced by British technology pioneer Kevin Ashton in 1999, envisioned a world where physical objects could be connected to the Internet via sensors, thereby enabling automatic identification and data collection without human intervention. Initially, the focus was on enhancing corporate supply chains through Radio-Frequency Identification (RFID) tags, illustrating the potential to streamline inventory management by tracking goods through Internet-Connected tags [55].

However, the vision is expanded beyond its initial focus. IoT represents the idea of enhancing connectivity and interaction among a wide range of devices, from consumer electronics to industrial equipment, through Internet Protocol (IP) technology. This enhanced connectivity facilitates the autonomous operation of devices, allowing them to perceive their environment, analyze data, and make decisions or communicate with other connected entities to optimize their function [42].

IoT applications cover a vast array of daily activities and industrial operations, characterized by the interaction between end devices and networking technologies. Two critical attributes of IoT systems are their heterogeneity and decentralization [53]. These two features make IoT devices a great candidate for decentralized computing and clustering algorithms. Integrating blockchain within the IoT structure can be utilized to ensure the security and integrity of data since the data stays immutable and can be tracked at any time through the blockchain network.

3.1. Blockchain Integration with IoT

The integration of blockchain technology with the IoT addresses security challenges heightened by the expanding IoT vision. As the number of IoT devices increases, they become prime targets for various cyber threats, including Distributed Denial-of-Service (DDoS) attacks, eavesdropping, and man-in-the-middle (MITM) attacks, among others [14, 52, 41]. The centralized nature of traditional IoT security frameworks, presents a single point of failure, exposing the network to additional risks concerning accessibility, authorization, and privacy [65].

Blockchain's decentralized architecture offers a robust solution to these vulnerabilities. By distributing validation across different nodes, blockchain reduces the risk of falsified validations and the feasibility of DDoS attacks through consensus mechanisms that apply transaction fees [30]. This decentralized approach not only enhances data integrity but also ensures that security measures are not solely dependent on third-party entities.

Blockchain can be used to enforce access control policies within IoT networks. Innovative blockchain-supported models have been proposed for improving network security by enabling access permissions through custom cryptocurrencies [73] or access control based on tokens where different roles can be assigned to users and control protocols [46]. These approaches, among others, demonstrate how blockchain can facilitate access management, prevent unauthorized access, and enhance the security of communications among IoT devices.

Furthermore, the immutable nature of blockchain allows the integrity of data and mitigates data integrity threats. For example, Boudguiga et al. proposed a mechanism to manage firmware updates in IoT devices, by using a consortium blockchain to update and record the firmware update in a peer-to-peer environment [15]. Additionally, the decentralized nature of blockchain can inherently improve the availability of services within IoT systems. Blockchain can be used to provide a resilient and fault-tolerant infrastructure against DDoS attacks [47].

3.2. Applications and Use Cases

Recently, there have been a lot of studies around applications and use cases of incorporating blockchain technologies into IoT systems [4]. These applications emphasize blockchain's built-in advantages, including its unchangeable nature, resilience to faults, ability to execute smart contracts, cryptographic protection, decentralized governance, and assurance of data integrity and authentication. It's noteworthy that certain applications opt in for specific blockchains designed specifically for their requirements, instead of relying on open-source options like Ethereum and Hyperledger Fabric [58].

A lot of Innovations in blockchain-based smart home applications have been proposed with a focus on enhancing security, privacy, and data integrity. Proposals range from decentralized key management systems for device interaction to Ethereum-based implementations for monitoring and accessing home appliances, emphasizing user privacy and system efficiency. Dorri et al. proposed a blockchain-based case study of a smart home. In this setting, the smart devices may communicate directly with each other or some external entities outside of the smart home. For instance, a motion sensor can detect some movement and signal the light bulb to turn on, facilitated by a miner that distributes a shared key for direct communication among devices. After a device receives and confirms the key as a valid key, it can then directly interact with other devices [21].

In the healthcare sector, IoT devices such as smartphones, wearables, or sensors implanted in patients are utilized in various ways, including early detection of medical issues, emergency alerts, and computer-assisted rehabilitation. These devices, alongside data collected from hospitals and diagnostic tools, enable automated healthcare management [62]. Therefore, it is important to develop a privacy-preserving IoT-based healthcare system to ensure the security and privacy of data. Blockchain can be used to address key challenges in healthcare data management, including data sharing, interoperability, and privacy. Projects like "MeDShare" [69] are proposed to solve the data sharing of medical data from different sources in a trust-less environment via blockchain. In the MeDShare project, the data can be traced and access to individuals in case of permission violation can be revoked [69].

3.3. Threshold Signatures For IoT Blockchain

In the landscape of blockchain-based applications for IoT devices, the concept of threshold signatures offers a robust solution for distributed trust and key management within IoT blockchain networks.

However, to the best of our knowledge, there are only a limited number of research explored threshold signatures for IoT blockchain ecosystems.

Yu et al. proposed STCChain, a Shamir's threshold cryptography approach that utilizes blockchain for industrial Internet of things (IIoT) key data. The problem is the limited storage capacity of IoT devices, data collected by these devices are usually stored in the cloud. However, the key used for data encryption or decryption is either directly stored on an IoT device which makes it vulnerable to side-channel attacks or managed by a third-party organization, which has security and privacy implications. To mitigate this issue, the private key is split between the IoT devices and the edge gateway using the Shamir secret sharing algorithm and encrypted using the public key of each device. Each encrypted key fragment is then published to the blockchain for storage [71].

Furthermore, Ricci et al. proposed a threshold scheme and benchmarked their solution on an ARM-based board to represent the IoT environment [54]. In section 4.6 we explain the proposed scheme and the result of the experiment in more detail.

Note that, the above-mentioned researches fall short of demonstrating and testing their solution through practical implementations in a real-world blockchain framework such as Hyperledger Fabric on IoT networks.

In contrast to the limited exploration of threshold signatures within blockchain-based IoT applications, the significance of our work lies in its practical integration with Hyperledger Fabric within IoT devices. This integration showcases the feasibility and benefits of threshold signatures, specifically FROST, in blockchain-based IoT networks.

4

Related Work

It is only recently that extensive research has begun to explore the concept of threshold signatures in the context of blockchain technologies. In evaluating related work on threshold signatures, we consider the following key criteria relevant for practical deployments:

- **Core Algorithm:** We examine the core algorithm used as the basis for the threshold signature scheme, specifically ECDSA or Schnorr.
- **Performance:** A threshold signature scheme is mostly affected by network latency. The number of communication rounds plays an important role in this regard. In each round of communication, a participant can either send a message to another participant or broadcast a message to every other participant in the group. We consider both the key generation and signing phase of the algorithm.
- **Majority Setting:** We analyze the assumption setting regarding the majority of participants. Two scenarios are considered: the honest majority assumption and the dishonest majority assumption. In the honest majority assumption, more than half of the participants are assumed to be honest and follow the protocol as intended. With a majority honest assumption in a (t, n) threshold setting, the supported value for t is $t \leq (n-1)/2$.
  On the other hand in the majority dishonest setting, it is assumed that up to $n-1$ out of n participants in a (n, n) threshold setting may behave maliciously or be compromised. For example, in a $(8, 10)$ threshold setup, the protocol must be secure even if up to 7 out of 10 participants are malicious.
- **Identifiable Aborts:** This criterion focuses on the ability to identify where a signatory member stops or deviates from the protocol. This can be useful to either punish the malicious user or just exclude them from the signing procedure.
- **Online Non-Interactivity:** It implies that after the message is known, each signatory can generate their individual partial signature without requiring interaction with any other signatory.

In this section, we discuss signature algorithms primarily implemented for blockchain and cryptocurrency applications, including one used in an IoT-blockchain-based system.

4.1. Lindell (2018)

Lindell and Nof [39] presented one of the earliest efficient protocols for providing threshold ECDSA. This protocol replaces Paillier additive homomorphic encryption with the in-the-exponent version of ElGamal encryption. It requires five rounds of communication for key generation and 8 for signing.

In the presence of malicious adversaries and static corruptions, the protocol adheres to a standard for instances where there is no honest majority, ensuring security with abort.

Lindell uses F_{mult} a helper functionality to perform multiplication functionality from additive shares. This helper function allows participants to securely multiply two values together.

Key Generation

Each participant P_i performs the following steps:

- P_i sends (init, G, q) to F_{mult} to run the initialization phase.
- P_i sends (input, sid_{gen}) to F_{mult} and receives (input, sid_{gen}, x_i).
- P_i waits to receive (input, 0) to F_{mult}.
- P_i sends (element - out, 0) to F_{mult}.
- P_i receives (element - out, 0, Q) from F_{mult}.
- Q is stored locally as the ECDSA public key.

Signing Protocol

Each participant performs the following steps upon input Sign(sid, m) where sid is a unique session id:

- P_i sends (input, sid | | 1) and (input, sid | | 2) to F_{mult} and receives (input, sid | | 1, k_i) and (input, sid| |2, p_i). The values k_i and p_i denote the current cumulative sum of k and p, respectively.
- After receiving (input, sid | | 1) and (input, sid | | 2) from F_{mult}, participant P_i sends (mult, sid | | 1, sid | | 2) and (element-out, sid | | 1) to F_{mult}.
- P_i receives (mult-out, sid | | 1, sid | | 2, r) and (element-out, sid | | 1, R) from F_{mult} where $r = k * p$ and $R = k * G$
- P_i computes $R = (r_x, r_y)$ and $r = r_x$ mod q.
- P_i sends (affine, 0, sid | | 3, r, m') to F_{mult}, where sid | | 3 will be associated with $m' + x * r$ mod q.
- P_i sends (mult, sid | | 2, sid | | 3) to F_{mult}.
- P_i receives (mult-out, sid | | 2, sid | | 3, b) from F_{mult}, where $b = p * (m' + x * r)$ mod q.
- P_i computes $s' = r^{-1} * b$ mod q and $s = min\{s, q, -s\}$.
- P_i outputs (r, s).

Identifiable Aborts

This protocol does not support identifying malicious parties.

Performance

The Paillier-based protocol was implemented in C++ and tested locally, without considering network latency, on a 2.3 GHz, 8-core Intel(R) i9. The key generation and signing protocol were tested ranging from 2 to 20 parties. With 2 parties, it takes 5 ms for key generation and 206 ms for the signing protocol. For 20 parties, it is 55 ms and 3675 ms for the key generation and signing protocol respectively.

The ultimate goal of Lindell's protocol was to showcase practical key generation, signing, and distribution for multiparty threshold ECDSA signature schemes. However, as an early pioneer, this protocol lacks some features such as identifying aborts and online non-interactivity."

4.2. GG20 (2020)

Gennaro and Goldfeder [27] improved the efficiency of GG18 [26] by reducing the number of communication rounds and new mechanisms to identify and deal with misbehaving parties who deviate from the protocol. Furthermore, the protocol assumes the majority of participants to be dishonest.

Key Generation

The key generation algorithm, based on a protocol similar to [26], is augmented to identify misbehaving parties. The protocol comprises three phases and 4 rounds of communications, including 3 rounds of communications for Feldman-VSS [9]:

• **phase 1:** Each player p_i selects a random value u_i from the set z_q. They compute $[KGC_i, KGD_i] = com(g^{u_i})$ using the commitment algorithm com. Finally, they broadcast kgc_i and the public key e_i for Paillier's cryptosystem.

• **phase 2:** Each player p_i broadcasts KGD_i, with y_i being the value decommitted by p_i. The player p_i performs a (t, n) feldman-vss of the value u_i, treating y_i as the "free term in the exponent". The public key is set to the product of all participants' public keys $y = \pi_i y_i$. Each player adds the private shares received during the n Feldman vss protocols. The resulting values x_i form a (t, n) Shamir's secret sharing [64] of the secret key $x = \sum_i u_i$.

• **phase 3:** Let $n_i = p_i q_i$ be the rsa modulus associated with ei. Each player pi proves in zero-knowledge (zk) that they know x_i using Schnorr's protocol [60].

Signing Protocol

The signing protocol, executed with the hash of the message m and the output of the previously described key generation protocol, unfolds in seven phases. Notably, the key generation protocol is a t-out-of-n protocol with the secret key x shared using (t, n) Shamir secret-sharing:

• **Phase 1:** Each Player P_i selects random values k_i, γ_i from Z_q. Compute $[C_i, D_i] = Com(g^{\gamma_i})$ using the commitment algorithm Com and broadcast C_i. Define $k = \sum k_i$ and $\gamma = \sum \gamma_i$.

• **Phase 2:** Each pair of players P_i, P_j engages in two multiplicative-to-additive share conversion subprotocols. Player P_i computes δ and σ a $(t, t+1)$ additive share of $k\gamma$ and kx respectively.

• **Phase 3:** Each player P_i broadcasts δ_i and collectively reconstructs $\delta = \sum_{i \in S} \delta_i = k\gamma$. Each player computes $T_i = g^{\sigma_i} h^{\ell_i}$ with $\ell_i \in Z_q$ proving in ZK that he knows σ_i and ℓ_i.

• **Phase 4:** Each Player P_i broadcasts the values decommitted by P_i namely, D_i, and $_i$. The player computes $R = g^{k^{-1}}$ and $r = H'(R)$.

• **Phase 5:** Each player P_i broadcasts $\bar{R}_i = R^{k_i}$ and a zero-knowledge proof of consistency between R_i and $E_i(k_i)$. If $g \neq \Pi_{i \in S} \bar{R}_i$, the protocol aborts.

• **Phase 6:** Each player P_i broadcasts $S_i = R^{\sigma_i}$ and a zero-knowledge proof of consistency between S_i and T_i. If $y \neq \Pi_{i \in S} S_i$ the protocol aborts.

• **Phase 7:** Each player P_i broadcasts $s_i = mk_i + r\sigma_i$ and sets $s = \sum s_i$. If the signature (r, s) is correct for m, the players accept, otherwise they abort.

Online Non-Interactivity

The GG20 protocol can be divided into an offline preprocessing stage and an online stage. During preprocessing, additive shares of s are derived from additive sharings of x and k. A distributed verification check is conducted on these shares of s in the preprocessing phase, eliminating the need for online interactivity. In the online phase, where the message m is known, players only require scalar multiplication and one communication round, resulting in lower communication costs.

Identifiable Aborts

The protocol distinguishes between aborts during the preprocessing and online stages. If an abort occurs, players reveal their random choices made during the protocol to verify behavior and identify bad players. Identifying misbehavior, even in cases where players slightly deviate from the protocol. Abort possibilities exist in both the KeyGen and Sign protocols, with the identification and removal of misbehaving players.

Performance

The GG20 was benchmarked locally, without considering network latency, using a 2018 Macbook Pro laptop with a 2.3 GHZ Intel Core i5 processor and 16 GB RAM. The signing protocol was tested ranging from about $500ms$ with 2 parties to $4000ms$ with 10 parties.

4.3. Damgard (2022)

Damgård et al. [18] introduced a schema with the assumption of an honest majority with n parties and a threshold condition of $t \leq (n-1)/2$. It uses a pre-processing phase to enable non-interactive signing operations. It requires 3 rounds of communication for the key generation algorithm and 4 rounds of communication for the signing algorithm, including 3 offline rounds that can be computed before the message is known as a pre-signature. Moreover, this protocol doesn't provide a fairness or termination guarantee. However, it can be extended with 2 additional pre-processing rounds to achieve fairness.

Computing Powers of a Point:

The $POWOPEN$ is a subprotocol to reveal the value $y = g^x$ given a sharing $[x]$ and a generator $g \in G$. The protocol is as follows:

• Each participant P_i sends $y_i = g^{x_i}$ to all other parties. Let f be the unique degree-t polynomial defined by the shares of the $t+1$ honest parties, where $f(0) = x$.

• Upon receiving all g^{x_j} for each $y_j \in \{y_{t+2}, y_{t+3}, ..., y_n\}$, P_i verifies the consistency of y_j with the degree-t polynomial defined by the initial $t+1$ values $y_1, y_2, ..., y_{t+1}$.

• If verification is successful, P_i concludes that $y_1, ..., y_{t+1}$ are valid points on f. The participant P_i then uses Lagrange interpolation "in the exponent" on $y_1, y_2, ..., y_{t+1}$ to compute $y = g^x = g^{f(0)}$.

Assuming that discrete logarithms in G are hard and the requirement $n \geq 2t+1$ holds, which means at least $t+1$ parties are honest. If any of the t corrupted parties attempt to cheat in the second step, all honest parties will abort.

Key Generation

The key generation algorithm aims to have the parties generate a sharing $[x]$ of a uniformly random value $x \in \mathbb{Z}_q$ and reveal to each participant $y = g^x$. To generate $[x]$, the parties run $[x] \leftarrow \text{RSS}(t)$ to obtain a sharing of a random value $x \in \mathbb{Z}_q$ over a random polynomial. To obtain $y = g^x$, the parties will run the protocol $y \leftarrow \text{POWOPEN}(g, [x])$.

Using the Shamir sharing protocol instead of verifiable secret sharing (VSS), enables the key generation protocol to abort if a malicious participant P_i causes $[x]$ to be an inconsistent share.

The protocol guarantees that if two parties output a public key, they output the same public key y. Note that no honest participant P_i reveals their value g^{x_i} until he has received shares x_j from all other parties P_j. This means that the corrupt parties P_j have to "commit" to their values $x^{(j)}$ before they see y. In addition, all subsets of $t + 1$ honest parties that receive output will receive shares of the same private key x satisfying $g^x = y$.

Signing Protocol

Assuming that the key generation has been done and all participants agree on the message m, the signing protocol is as follows:

• Each participant generates $[a]$ using Shamir sharing and then compute $[w] = [a][k]$ and open $[w]$ using $POWOPEN$.

• To ensure the correctness of $[w]$, compute an authenticator $W = g^{ak}$ using $POWOPEN$ with g^k as the base.

• Compute the signature value $[s] = [k^{-1}](m + r[x])$, where m is the (hashed) message, and $r = F(g^k)$.

• Verify the resulting signature (r, s) on the message m using the public key y.

Online Non-Interactivity

In the basic variant of this algorithm, after the participants receive the message m, the sharing of the signature can be computed using only local operations without interacting with other participants.

In the online phase, the protocol lacks fairness termination guarantees. Fairness means that the adversary could potentially see the signature (r, s) and abort the protocol before other honest parties receive the signature. Note that this does not count as forgery since it happens with messages the honest parties intended to sign. However, participants may retry on abort and the adversary could end up with multiple valid signatures on the same message m without the honest parties being aware. The protocol can achieve fairness by extending the pre-processing with 2 additional rounds.

Identifiable Aborts

This protocol doesn't support identifying malicious parties.

Performance

The protocol was benchmarked in both LAN and WAN settings using a client-server setup. Each participant uses one CPU core to execute the protocol.

In the LAN setup, the key generation algorithm took $40ms$ with 2 participants and $66ms$ with 4 participants.The signing protocol took $54ms$ for 2 participants and $109ms$ with 4 participants. In the WAN setup, the key generation algorithm took $1.22s$ with 2 participants and $1.48s$ with 4 participants.The signing protocol took $2.2s$ for 2 participants and $2.7s$ with 4 participants.

4.4. DKLS (2019)

Doerner et al. [20] is the only threshold scheme in this section with a non-constant number of signing rounds i.e., $6 + \log t$. This schema is only non-interactive in the pre-processing phases and requires one round of online activity once the message is known. This protocol assumes the majority of participants to be dishonest.

Key Generation

This protocol, parameterized by the party count n, threshold size t, and elliptic curve (G, G, q), utilizes functionalities for two-party multiplication (2PMul), modular inverse sampling (ModInv), and committed zero-knowledge proofs for discrete logarithms (ZKP_DL). The protocol involves public key generation where each party P_i samples a random polynomial point $p(i)$ of degree $t - 1$. Interactions between parties result in each party computing a point on the polynomial p. The joint public key is computed using zero-knowledge proofs for commitment.
• Each party P_i samples a random polynomial p_i of degree $t - 1$.
• For all pairs of parties, they exchange polynomial evaluations with other parties
• Each party P_i computes its point: $p(i) = \sum_{j\in[n]} p_j(i)$
• Each party P_i computes the commitment $T_i = p(i) \cdot G$ and sends to other participants
• Each party P_i receives T_j for each $j \in [n]$. Abort the protocol if the ZKP_DL fails
• The parties compute the shared public key using any subset $J \subseteq [n]$ with $|J| = t$:
$pk = \sum_{j\in J} \lambda_j^J(0) \cdot T_j$ where $\lambda_j^J(y)$ is party P_i's Lagrange coefficients for interpolating p at location y

Signing Protocol

This signing protocol is parameterized by the party count n, threshold size t, a subset of parties $P \subseteq [n]$, elliptic curve (G, G, q), and a statistical security parameter s. It relies on functionalities such as two-party multiplication (2PMul), modular inverse sampling (ModInv), and commitment (FC). Each party in the subset P generates a signature σ for a given message m.
• Each party P_i invokes t-party Modular Inverse Sampling to obtain (u_i, v_i, R)
• Each party P_i computes its Lagrange coefficient and the additive share of the secret key sk_i.
• Each pair of parties P_i and P_j performs Two-party multiplication (2PMul) with input $\{sk_i, v_i\}$ and $\{v_j, sk_j\}$ and receive output shares $\{w_i^{j,1}, w_i^{j,2}\}$ and $\{w_j^{i,1}, w_j^{i,2}\}$ respectively.
• Each party P_i computes $w_i = sk_i \cdot v_i + \sum_{j\in P}(w_i^{j,1}, w_i^{j,2})$
• Each party P_i computes $\Gamma_i^2 = v_i \cdot pk - w_i \cdot G$ and $\Gamma_i^3 = w_i \cdot R$ and commits both values
• Upon receiving commitments from all other parties, each P_i collects other parties's decommitments and aborts if the sum of all is not equal to zero or pk
• Each party P_i calculates $sig_i = H(m) \cdot v_i + r_x \cdot w_i$ and broadcasts sig_i
• Each party aggregates the partial signatures to calculate the message signature.

Online Non-Interactivity

In the Sign protocol, the only element dependent on the message m is the final protocol message generated which is computed by sig_i and broadcasted. The reconstruction of the signature, performed in the last step using all parties' final messages, requires knowledge of r_x, pk, and the messages themselves. Consequently, the protocol is non-interactive in the preprocessing model. If parties preprocess the protocol up to Step 7 before m is known, completing the protocol (online phase) requires only one round. In this round, parties simultaneously transmit one Z_q element each to those requiring the output. This approach is nearly optimal in terms of communication cost, with the overhead being proportional to the ratio between $q =$ and the security level of the elliptic curve (G, G, q) in bits.

Identifiable Aborts

This protocol doesn't support identifying malicious parties.

Performance

The protocol was benchmarked in both LAN and WAN settings. In the LAN setting, a set of 256 nodes in Google's South Carolina data center was used where all parties resided on individual machines in the same data center. The latency was on the order of a few tenths of a millisecond. The number of parties (n) ranges from 2 to 256. The key generation took from $60ms$ to $2000ms$ and the signing algorithm took from $500ms$ to $16000ms$.

Furthermore, a set of benchmarks was performed on a group of 3 Raspberry Pi model $3B+$, quad-core ARM processor clocked at $1.4GHz$, to demonstrate the performance of the protocol on small low-powered devices. In a 3-out-of-3 setting, the key generation algorithm took $1390ms$ and the signing algorithm took $93ms$.

4.5. CMP (2020)

Canetti et al. [16] introduced a threshold scheme with two protocols. The protocols are distinguished by the number of communication rounds and the identification process for detecting malicious parties.

In the first protocol, the signing protocol takes 4 rounds but the identification process is less efficient and is quadratic in the number of participants (with $O(n^2)$ complexity). In the second protocol, the identification process is more efficient i.e., it is only linear in the number of parties (with $O(n)$ complexity) but the signing protocol takes 7 rounds of communication.

Each protocol offers two variants: one for online signing and one for non-interactive signing. The online variant requires the participants to do the pre-signing and signing phase for each new signature, while the offline variant allows for pre-signing before the message is known. Further, this protocol employs zero-knowledge proofs, which introduce an efficiency bottleneck.

Key Generation

The key generation phase involves each party $P_i \in \mathcal{P}$ sampling $x_i \leftarrow F_q$ and sending the public-key share $X_i = g^{x_i}$ to all other parties. Additionally, each party provides a Schnorr proof of knowledge for the exponent. The public key is then computed as $X = \prod_j X_j$. To enhance malicious security, parties commit to their public-key share X_i and the first message A_i of the Schnorr proof using an oracle. This prevents the adversary from influencing the private-key distribution and commits the adversary to the first Schnorr proof message for later witness extraction in the reduction process.

Upon receiving all relevant values and detecting no inconsistencies, the final public key X is computed as the product of all public key shares X_j, and the secret key shares x_i along with the public key shares X are stored.

Signing Protocol

For the signing protocol, we only look at the version with 4 rounds of communication (3 rounds of pre-signing plus 1 round of online signing).

Presigning: In the three-round version of the pre-signing protocol, parties are instructed to send their Γ_i's in round two (with the relevant proof) and then compute and send $\Delta_i = \Gamma^{k_i}$ in round three (with the relevant proof).

Identification Process: In this version of the protocol, the identification process is conceptually simpler but computationally more demanding. In case of failure, the parties proceed as follows:

1. For each $j \neq i$, reprove that $\{D_{j,i}\}_{j \neq i}$ are well-formed according to $\Pi_{\ell}^{\text{aff-p}}$, for $\ell \neq j$.
2. Compute $H_i = \text{enc}_i(k_i \cdot \gamma_i)$ and prove in zero knowledge that H_i is well-formed with respect to K_i and G_i.
3. Prove in zero knowledge that δ_i is the plaintext value modulo q of the ciphertext obtained as $H_i \cdot \prod_{j \neq i} \cdot D_{i,j} \cdot F_{j,i}$.

Signing: Once the (hash of the) message m is known, on input $(\text{sign}, \ell, i, m)$ for the ℓ-th revealed point on the curve, the signing boils down to retrieving the relevant data and computing the right signature share. Namely, retrieve (ℓ, R, k, χ), compute $r = R|_{x\text{-axis}}$, and send $\sigma_i = km + r\chi \mod q$ to all. Erase the tuple (ℓ, R, k, χ).

Online Non-Interactivity

To sign non-interactively, the parties need to prepare some number of pre-signatures in an offline stage. For a pre-signing parameter $L \in \mathbb{N}$, the parties run the pre-signing phase L times concurrently and obtain pre-signing data. Later, for each signature request using pre-signing data $(l, R_l, k_l^i, \chi_l^i)$ and message `msg`, the parties run the signing phase for the relevant input to generate a signature. The parties then erase the pre-signing tuple $(l, \ldots)$. It is important to ensure that, as part of the refresh stage, any unused pre-signatures are discarded. Alternatively, it is possible to keep the pre-signing data as long as it is appropriately refreshed.

Identifying Aborts

The identification of corrupted parties in case of non-malicious failures involves two distinct processes. The first, less efficient method incurs an $O(n^2)$ penalty and requires proving in zero-knowledge that the transmitted values were consistent with the entire transcript when nonces or signature strings were malformed. The second, more efficient method, incurs three extra rounds during the pre-signing stage but only requires $O(n)$ computation. It involves opening relevant ciphertexts to check if the right message was sent, allowing parties to identify the culprit without revealing the master secret key and without the need to store the entire transcript for the online phase. The challenge lies in verifying the well-formedness of pseudo-keys, specifically $\bar{R}_i$ and Y_i, which are essential for signature-share verification.

4.6. RDCC (2022)

Ricci et al. proposed a Schnorr-based threshold signature aiming to be applied to Blockchain technologies [54]. The key generation (setup) algorithm involves collaboration among n signers to establish a polynomial $f(x)$ of degree $t-1$, with $sk = \sum k_i$ as the constant term, where sk is the secret key used in the signature. The setup procedure consists of two rounds of communication. The signing algorithm is divided into two phases and requires three rounds of communication between the main device and the secondary devices. In the last round, the main device aggregates all signature fragments and produces a Schnorr signature. This protocol assumes that the majority of participants are dishonest. Furthermore, the algorithm does not support Identifiable Aborts and Online Non-Interactivity.

In this signature protocol, there are two main entities: the Signer, responsible for generating signatures collaboratively with co-signers, and the Verifier, a Blockchain node that receives and validates user transactions. The Signer operates with a Main Device (MD) for signing transactions and Secondary Devices (SD) for co-signing. The Verifier ensures the validity of signatures in Blockchain transactions.

Let D_j be the signer's device performing the protocol. Each D_j owns a secret key share k_j where $j = 1, \ldots, n$.

Key Generation

The scheme is based on Shamir protocol [64] and, therefore, requires that n signers agree on a polynomial $f(x)$ of degree $t-1$ that has $sk = \sum k_i$ as the constant term. Note that sk is the secret key used in the signature. The Setup algorithm consists of two phases:
1. $(\Lambda_j, pk, pk_j, pk_{p,j}) \longleftarrow$ **ParGen** $\longleftarrow (n, t, \kappa)$: each device D_j for $j = 1, \ldots, n$ does as follows and with respect to a security parameter κ:
• Generate at random $d_1^{(j)}, \ldots, d_{t-1}^{(j)}$,
• Generate the Paillier's key pair $(pk_{p,j}, sk_{p,j})$,
• Generate at random $k_j \in \mathbb{Z}_{q_{EC}}$,
• Compute $pk_j = g^{k_j}$.
The values $\Lambda_j = (k_j, d_1^{(j)}, \ldots, d_{t-1}^{(j)}, sk_{p,j})$ are privately stored in each device, whereas $(pk_j, pk_{p,j})$ are made public. An agreed user computes the common key $pk = \prod_{i=1}^{n} pk_i$ (i.e., Blockchain wallet public key).
2. $(\alpha_j, f(\alpha_j)) \longleftarrow$ **PolyEval** $\longleftarrow (\Lambda_j, pkp, j)$ The algorithm outputs for each device D_j the pair $(\alpha_j, f(\alpha_j))$ where $\alpha_j = j$ is publicly known, and $f(\alpha_j)$ is kept secret.

Signing Protocol

The Signing algorithm consists of two phases:
Session Key: For each device D_j in the set of signers, compute the session key s_j and store it privately:

$$s_j = f(\alpha_j) * \Pi_m \frac{\alpha_m}{\alpha_m - \alpha_j} \bmod qEC$$

where m is the other signer's index.
Signature: Each device D_j commits to its random value r_j and sends the commitments c_j to the Main Device (MD). MD aggregates the commitments to receive the common commitment c. Then, c is sent back to the Secondary Devices (SDs) along with the signing message m (Blockchain transaction). Each D_j generates its signature fragment (z_j) on m. Finally, MD aggregates all signature fragments to produce the Schnorr signature $\sigma = (e, z)$ on the Blockchain transaction, which is sent for validation.

Performance

In the experimental evaluation, one single Raspberry Pi 4 with 4GB of RAM represented the Signer MD and SDs devices, and the Verifier's device. The testing program was written in C. The benchmark revealed that the Key Generation algorithm was the most time-consuming, taking around 1 second for $(t = 5, n = 5)$ and about 4 seconds for a $(t = 10, n = 9)$ setting. The signing algorithm required about $10ms$. Communication overhead was not considered in the above measurements.

4.7. Robustness in FROST

In the context of cryptographic protocols, particularly those involving multiple signers, the concept of robustness is defined by the ability of a signing protocol to successfully generate a valid signature with the participation of up to n signers, despite the presence of a dishonest majority of $(t - 1 \geq n/2)$ attempting to obstruct the process. This definition underscores a scenario where the protocol remains operational and achieves its goal of producing a valid signature, as long as there are t honest signers present, even if the rest (up to $n - t$) are acting maliciously to prevent the completion of the signing process. In instances where $t1$ signers are malicious, the number of remaining honest signers ($n(t1)$) may fall below the threshold needed to generate a signature, illustrating a practical limitation in the protocol's ability to guarantee liveness in all circumstances where unforgeability is still assured [54].

González et al. [29] introduces some improvements to the original FROST scheme. The proposed protocol achieves the ability to identify cheating participants in the key generation phase, allowing the participants to check the validity of complaints against potentially cheating participants, leading to the exclusion of cheaters without aborting the key generation protocol. This will ensure robustness in the key generation algorithm.

Additionally, [29] presents a static public key for the FROST allowing regular key redistribution between participants while keeping the resulting distributively generated key unchanged. The group's established public and private keys remain unchanged for the lifetime of signers while only the signing shares of each participant are updated over time. This approach facilitates the verification process of the generated threshold signature, requiring signers to communicate their public key to the verifier only once during the group's lifetime. The set of participants can also be updated while the key would remain unchanged.

Furthermore, the ROAST protocol [56] addresses the robustness limitation of the FROST signing protocol. This protocol ensures that a signing session involving up to n signers will succeed and produce a valid signature when t honest signers are present, even if all remaining signers in the session are malicious. ROAST, acting as a wrapper for FROST or any other threshold signature that turns them into a scheme with robust and asynchronous signing protocol that as long as the underlying signing protocol meets three conditions: has one preprocessing round and one signing round, provides identifiable aborts, and is unforgeable under concurrent signing sessions.

4.8. Summary

Table 4.1 summarizes our contribution of providing the literature survey of the threshold signatures in the blockchain context, highlighting their features and providing their fair comparison.

Among the various threshold signature schemes discussed above, FROST offers a compelling choice for our work. Built upon the Schnorr signature algorithm, FROST demonstrates efficient performance with only two rounds of communication for both key generation and the signing protocol. Its potential optimization to a single round signing through preprocessing aligns well with our goal of minimizing communication overhead. Minimizing communication overhead is essential in IoT device interactions, as energy consumption due to radio module activation and delays involved in different communication protocols such as BLE, Wi-Fi, LoRaWAN strongly influence the device's overall energy consumption.

While FROST mainly emphasizes network efficiency, it also includes a mechanism for detecting misbehavior, such as when a participant provides a malformed share. In such cases, the remaining honest participants can identify the misbehavior, abort the protocol, and take corrective actions, such as excluding the misbehaving participant from subsequent rounds, although this may require restarting the protocol [35]. These qualities make FROST a strong candidate for the efficient utilization of Internet of Things (IoT) devices in blockchain applications, which is central to our research focus. It is worth noting that the protocols [56, 29] utilize FROST as a main component, with reasonable overhead in terms of communication rounds. Given that one of our primary objectives is to develop an efficient blockchain-based IoT-based threshold signatures protocol we select FROST signature as the core element

Protocol	Core algorithm	Performance	Majority	Identifiable Aborts	Online Non Interactivity
Lindell (2018) [39]	ECDSA	Key Gen: 5 rounds Signing: 8 rounds	Dishonest	✗	✗
GG20 (2020) [27]	ECDSA	Key Gen: 4 rounds Signing: 7 rounds	Dishonest	✓	✓
Damgard (2022) [18]	ECDSA	Key Gen: 3 rounds Signing: 4 rounds	Honest $t \leq (n-1)/2$	✗	✓
DKLS (2019) [20]	ECDSA	Key Gen: 8 rounds Signing: $6 + \log(t)$	Dishonest	✗	✓
CMP (2020) [16]	ECDSA	Key Gen: 3 rounds Key Refresh: 3 rounds Signing: 4 rounds	Dishonest	✓	✓
RDCC (2022) [54]	Schnorr	Key Gen: 2 rounds Signing: 3 rounds	Dishonest	✗	✗
ROAST (2022) [56]	Schnorr	Key Gen: N/A Signing: > 3 rounds	Dishonest	✓	✗
ICE-Frost (2023) [29]	Schnorr	Key Gen: 2 rounds Complaint Management: 2 rounds Signing: 2 rounds	Dishonest	✓	✓
FROST (2020) [35]	Schnorr	Key Gen: 2 rounds Signing: 2 rounds	Dishonest	✓	✓

Table 4.1: Summary of threshold signatures and their characteristics

of our proposed protocol. Furthermore, we assume that as our proposed solution is designed to work with FROST, the aforementioned FROST-based protocols will also be compatible with our solution, but a reasonably higher latency might occur.

5

Proposed Protocol

In this chapter, we specify the setup in which our IoT-enabled FROST signing operates. This setup functions in conjunction with the Hyperledger Fabric blockchain network. We also elaborate on our implementation to enable registration and enrollment for a new group of signer devices in the Hyperledger Fabric network, particularly when the registration and enrollment procedure involves the Fabric Certificate Authority. Finally, we demonstrate the entire transaction submission process, covering transaction generation and signing by the devices using FROST and its submission to the blockchain network.

5.1. Setup of the network architecture

In our proposed protocol's setup, we consider the following scenario:

- There exist n users, denoted as $u_1, u_2, \ldots, u_n$, who have already used the FROST algorithm to establish a group G. Each user possesses the group's public key pk_g and their partial secret key sk_i, all obtained from the FROST distributed key generation algorithm.
- The group G is configured with a threshold of t where $2 \leq t \leq n$. This threshold specifies the minimum number of participants required to collaboratively sign a transaction.
- A secure communication channel exists among the users that allows for broadcasting or sending a message between any two users.
- Regardless of which user within the group G initiates a registration process or a transaction submission, no user has any power over other participants within the group.
- There exists a channel set up and running on the Fabric network and Fabric-CA is used to register, enroll and generate the certificates for clients.

5.2. Registering and Enrolling with a Certificate Authority

In this section, we describe the registration and enrollment process of a new group identity corresponding to the group of IoT devices using FROST.

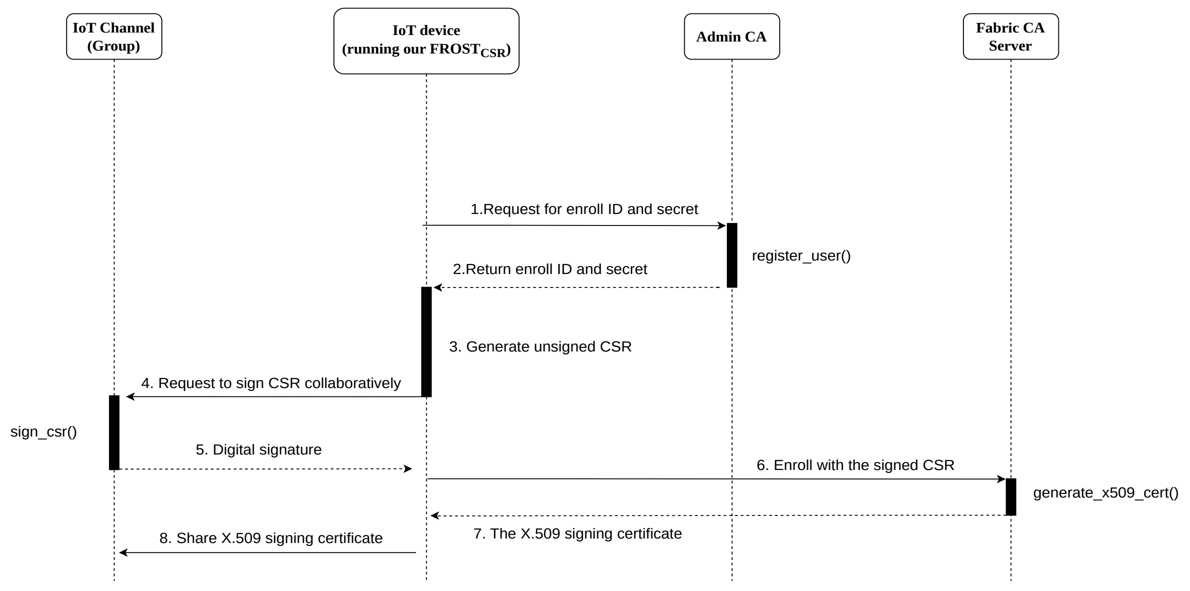

IoT Channel (Group)
IoT device (running our FROST$_{CSR}$)
Admin CA
Fabric CA Server
1.Request for enroll ID and secret
register_user()
2.Return enroll ID and secret
3. Generate unsigned CSR
4. Request to sign CSR collaboratively
sign_csr()
5. Digital signature
6. Enroll with the signed CSR
generate_x509_cert()
7. The X.509 signing certificate
8. Share X.509 signing certificate

Figure 5.1: The registration and enrollment flow

In Section 2 we described the generic procedure of registration and enrollment of a new identity via Hyperledger Fabric Certificate Authority (CA). In the default procedure, Fabric CA generates the private-public key pair for a new user. However, in our approach, using Fabric CA to generate the private-public key pair of the group G is not feasible. In the FROST signature protocol, each signer possesses a share of the group's private key, which is collectively generated by each participant of the signatory group. Additional details about the FROST key generation algorithm can be found in Section 2. Therefore, the recommended approach is as follows: Initially, a user initiates the registration process by requesting an enroll ID and the corresponding secret from the admin of a CA. In the next step, the user uses their enroll ID and public key to create a Certificate Signing Request (CSR) and sign it using their private key. It is important to note that with this approach the private key will never be shared with the admin CA.

The main challenge in threshold signature scenarios is sending the CSR to the Fabric CA without disclosing the group's shared secret key. In the current version of Fabric SDK (V2.5), the private key of the user is used to generate and sign the CSR and finalize the enrollment process. The CSR contains three main components [44]: *Certificate Request Information:* This section contains the enroll ID, public key, host, serial number, and other relevant user information. *Signature Algorithm:* The signature algorithm identifier e.g, ECDSA, ED25519, etc. *Digital Signature:* The signature on the certificate request information which is signed by the user's private key.

To address the aforementioned challenge, we have developed an API named FROST$_{\text{CSR}}$ in Go on top of the Fabric SDK core. This API enables one of the users within the group to generate an unsigned CSR document without requiring the private key.

In our proposed protocol which is depicted in Figure 5.1, the registration and enrollment process begins with a user delegated by group of users, G, and this user is denoted as u_d. The delegated user runs our FROST$_{\text{CSR}}$ API, which initiates the enrollment by sending a registration request to the admin of the CA. The CA admin assigns a unique enrollment ID and a corresponding secret to the user. Using the enroll ID and group's public key, the user generates an unsigned CSR. The CSR contains information such as the enroll ID, the group's public key, and the signature algorithm identifier i.e., ED25519. The

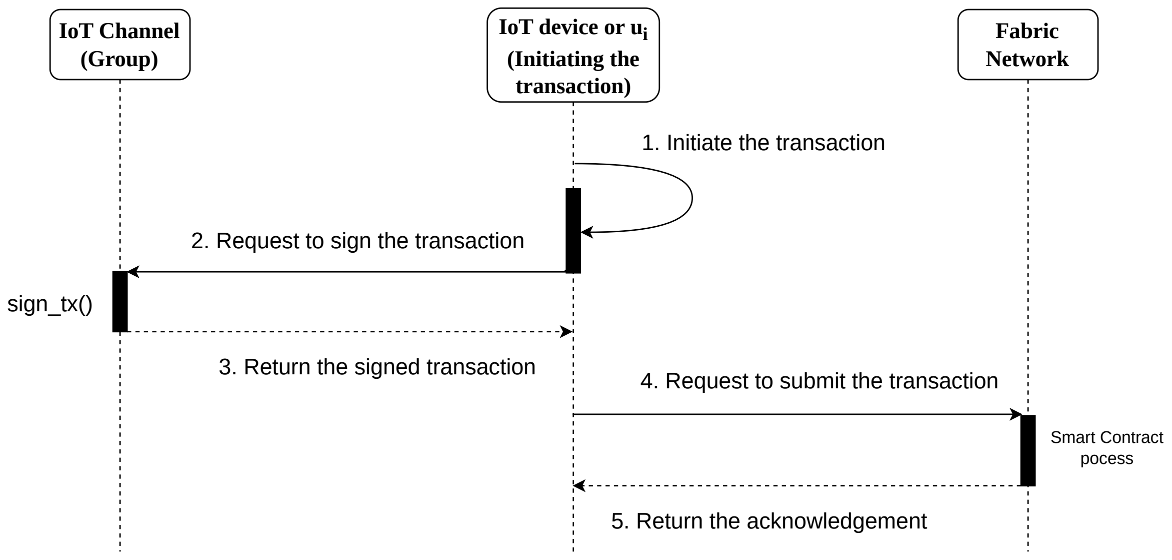

IoT Channel (Group)
IoT device or u_i (Initiating the transaction)
Fabric Network
1. Initiate the transaction
2. Request to sign the transaction
sign_tx()
3. Return the signed transaction
4. Request to submit the transaction
Smart Contract pocess
5. Return the acknowledgement

Figure 5.2: The transaction flow

CSR is sent to the other signers within the group to collaboratively sign it. Note after signing the CSR, the user u_d can not change the content of the CSR since it causes the verification algorithm to fail.

Finally, the user u_d sends the signed CSR to the Fabric CA server to complete the enrollment. Once the Fabric CA verifies the signature, it will generate an X.509 sign certificate and return it to the user. The user u_d can broadcast the signed certificate to the other members. Note that possession of a signed certificate by a user is not sufficient to submit a transaction. The transaction still needs to be signed by at least t members of the group.

5.3. Transaction generation and signing API

In the context of our proposed protocol depicted in Figure 5.1, consider a scenario where the user u_i proposes a transaction to be submitted to the Fabric network. Since no one in the group G has access to the private key required for message signing, a customized signing procedure must be employed within the Fabric gateway interface to facilitate transaction submission. Fabric SDK includes an interface called Gateway, which provides tools for interacting with the blockchain network. In this interface, the client can submit a custom signing message which in our protocol is replaced by the Frost sign algorithm. User u_i sends the unsigned transaction proposal to the other participants within the group. This allows other participants to review and approve the transaction. The transaction proposal requires to be approved, and thus signed by at least t participants. After, obtaining the aggregated signature, user u_i is authorized to submit the transaction proposal to the Hyperledger Fabric network. Figure 5.2 shows the transaction flow.

In conclusion, we can affirm that the registration and enrollment is enabled with our proposed $\text{FROST}_{\text{CSR}}$ API, alongside with the modified Fabric SDK enabling the signature proposal creation and its signature via FROST i.e., a group of users/IoT sign the transaction proposal. Our solution therefore does not require the modification of Hyperledger Fabric v3.0 core functionalities/source codes.

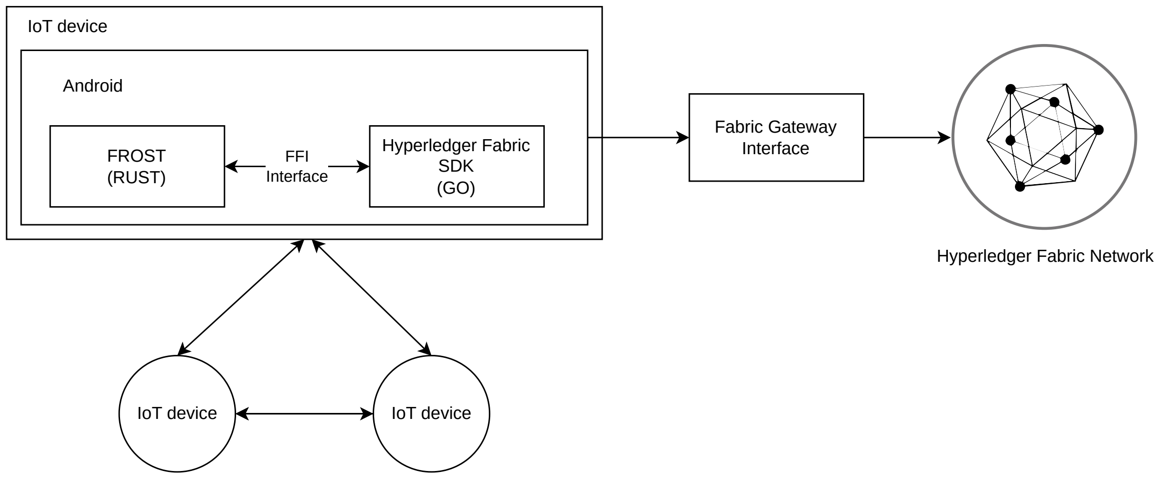


Figure 5.3: The Architecture Overview

5.4. Technical details about the APIs

The first component of our system is the FROST implementation in Rust language, which builds upon the ZF FROST library to support core FROST key generation and signing algorithms. Specifically, we used the ED25519 curve variation for generating signatures, ensuring compatibility with the standard ED25519 signature verification algorithm. While the core ZF FROST library lacks native support for communication channels, we extended its capabilities in our implementation. In our work, we eliminated the role of the signature aggregator (SA) and replaced it with a broadcast communication to simplify the coordination process [35]. To enable communication between the participants, we developed a Transmission Control Protocol (TCP) layer on top of the ZF FROST library using data serialization. In this setup, devices can either broadcast messages to all connected peers or direct messages to a specific device within the group. Given that all devices operate on the local network environment, we omitted the Transport Layer Security (TLS) protocol in this communication channel.

Furthermore, we implemented FROST$_{\text{CSR}}$ in the Go programming language to generate unsigned CSR which later will be sent to other participants to be collaboratively signed. The signed CSR is used by Fabric CA to enroll identities within the Hyperledger Fabric Network. Additionally, for transaction submission, we leveraged the Fabric Gateway 1.3.2 interface that is part of the Fabric SDK. This interface allows users to use custom signing routines for signing transactions. In our case, we integrated the FROST implementation in Rust to be used in the Fabric Gateway interface using the Foreign Function Interface (FFI) feature of Rust. FFI acts like a bridge that enables interoperability between different programming languages. Specifically, it allows the Go language to call a function written in Rust through a shared library or module. FFI allows for the exchange of data between the two languages. In our implementation, we used raw pointers to byte arrays to pass messages from Go routine to the FROST signing routine in Rust and return the signatures generated by FROST to the Go routine.

Finally, we used a terminal emulator on Android to compile and execute both our FROST implementation and FROST$_{CSR}$ natively on the Android platform. This approach minimized overhead and enabled us to efficiently run and test our API implementation. Figure 5.3 shows an overview of all components used in the implementation. Note that all IoT devices follow the same structure. Further, in Figure 5.3 we used Android as an example of an IoT device.

6 Experiments And Results

In this section, we provide details about two experiments conducted to evaluate the performance and feasibility of our proposed approach.

6.1. Experiment 1: FROST in IoT devices

In the initial experiment, our focus is on conducting a comparative analysis of FROST alongside two other popular threshold signature schemes namely, GG20 [27] and DKLS [20]. The primary objectives of this experiment are the evaluation of performance metrics and the measurement of energy consumption of the three algorithms when executed on resource-constrained IoT devices.

6.1.1. Setup

Hardware: We used four Raspberry Pi 2 devices, each equipped with a 900MHz Cortex-A7 CPU and 1 GB of RAM. These devices were running Arch Linux for ARM with Linux kernel version 6.1. Further, we used the OTII Arc Pro power meter device to measure the energy consumption of the Raspberry PI while running each algorithm. The power meter has a sample rate of $50ksps$.
Network Environment: To ensure optimal network conditions for our experiment, we conducted our experiment on a 1Gbps LAN network. The four Raspberry Pi devices are connected to the network via an Ethernet cable.
Software: For the core implementation of the FROST key generation and signing algorithm, we used ZF FROST library version 0.7.0 which is implemented in Rust. However, for our experiments, we need to run both key generation and signing algorithms distributively on each device. Further, we decided to avoid the application of the signing aggregator. Therefore, we extended the ZF FROST library to add support for both requirements: the distributed key generation/signing algorithm and signing aggregator removal. More details in section 5.4. Furthermore, to measure the energy consumption we used the OTII 3 app.

6.1.2. Methodology

To measure the energy consumption, one of the Raspberry PIs was connected to the power meter device. The power meter device provided precise control over the voltage supplied to the Raspberry PI while

Scheme	Signing	Energy Consumption
FROST	**57 ms**	**0.09 J**
DKLS	147 ms	0.26 J
GG20	23.3 sec	36.8 J

Table 6.1: Energy Consumption of various threshold signature with 4 participants

simultaneously measuring the voltage and current consumed by the device. This setup allows for accurate monitoring of energy consumption during the execution of the algorithms.

To mark the precise start and end times of the algorithms, a GPIO pin on the Raspberry PI was connected to the GPIO interface of the OTII device. Sending signals through the GPIO pin from the Raspberry PI to the power meter device enabled precise measurement of the algorithm's runtime phase. The device measures one sample of current and voltage per microsecond.

The energy consumption is calculated by multiplying the average power, derived from the current and voltage measurements, by the total time taken to execute the algorithm.

6.1.3. Results

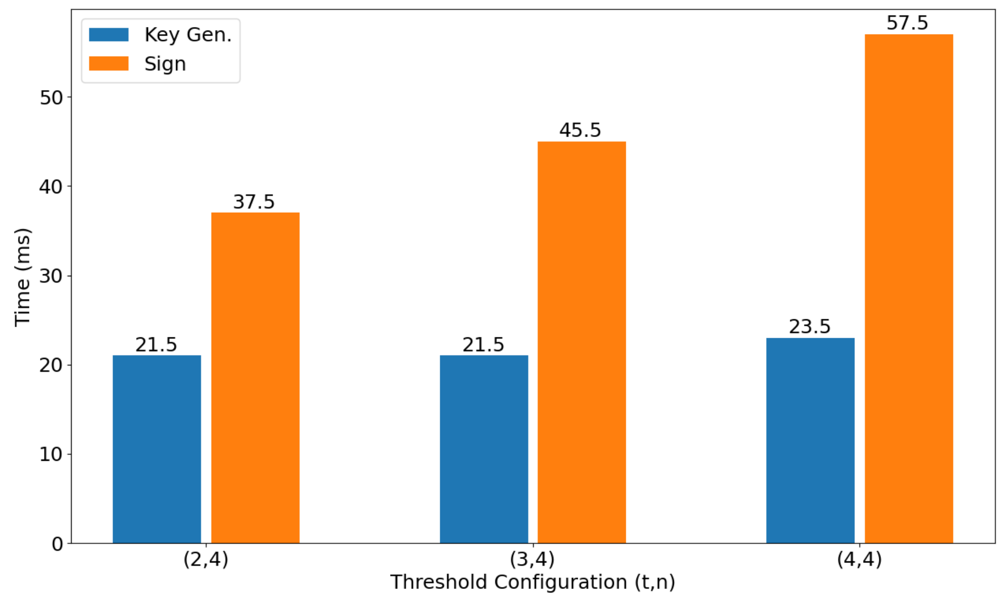


Figure 6.1: Running times of FROST

The results of the first experiment, as depicted in Figure 6.1, showcase the performance of both key generation and signing algorithms across varying threshold configurations. The key generation algorithm exhibits a relatively consistent performance across different threshold configurations and the number of devices, with an average execution time of approximately 22 milliseconds. In contrast, the signing algorithm demonstrates a linear relationship with the number of devices participating in the signing protocol, with the execution time increasing proportionally to the number of devices.

Table 6.1 presents a comparison of the energy consumption of the three threshold signature schemes with four participants. Notably, FROST demonstrates significantly lower energy consumption compared

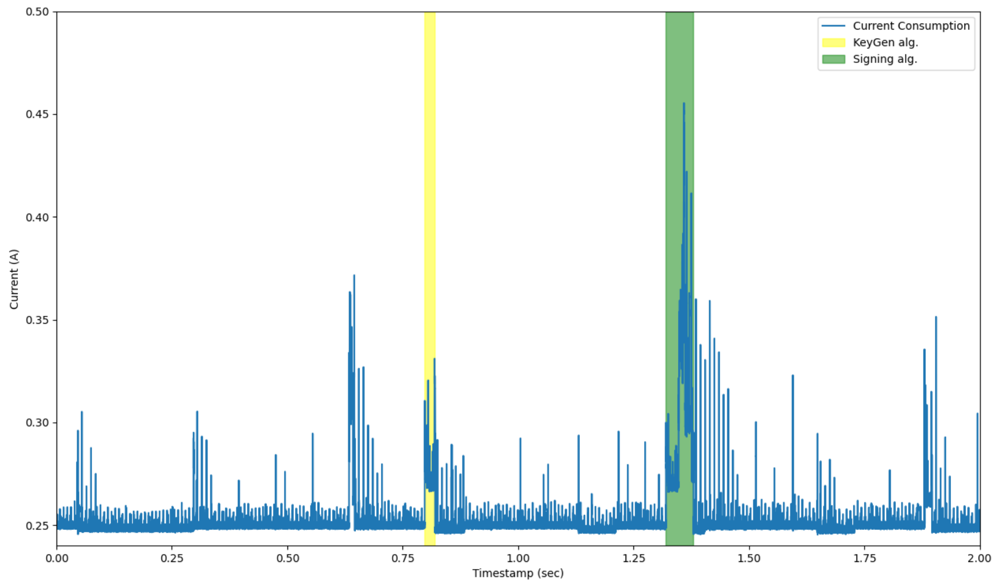


Figure 6.2: Current Consumption of FROST

to DKLS and GG20. Specifically, FROST exhibits a runtime of 57 milliseconds for signing, consuming only 0.09 joules of energy. In contrast, DKLS requires 147 milliseconds and consumes 0.26 joules, while GG20's signing algorithm takes 23.3 seconds and consumes a substantial 36.8 joules of energy.

The comparison highlights that FROST takes approximately three times less time than DKLS and about 400 times less time compared to GG20. This significant reduction in execution time translates into lower energy consumption over time, making FROST a more efficient choice for IoT applications.

Additionally, Figure 6.2, Figure 6.3, and Figure 6.4 provide a visual representation of the current consumption of all three algorithms during both key generation and signing phases. These figures offer insights into the energy consumption patterns of each algorithm, further confirming the efficiency of FROST in IoT environments.

Overall, the results confirm that FROST is a feasible choice for deployment in IoT applications, offering a balance of performance and energy efficiency.

6.2. Experiment 2: Integration of FROST with Hyperledger Fabric

In the second experiment, we focused on integrating FROST with the Hyperledger Fabric blockchain network to assess its performance in the context of transaction submissions by IoT devices.

6.2.1. Setup

Hardware: We used the four Raspberry Pi 2 devices similar to subsection 6.1.1. Additionally, we used an Android smartphone with an octa-core CPU clocked at 3.19 GHz (Cotex-X2 + Cortex-A710 + Cortex-A510) and 12 GB of RAM. The smartphone was running Android version 13 with Linux kernel 5.10. To ensure consistency and avoid performance hit, all the experiment codes have been compiled and executed natively on the respective devices.

Network Environment: In addition to the network configuration of subsection 6.1.1, the Android device

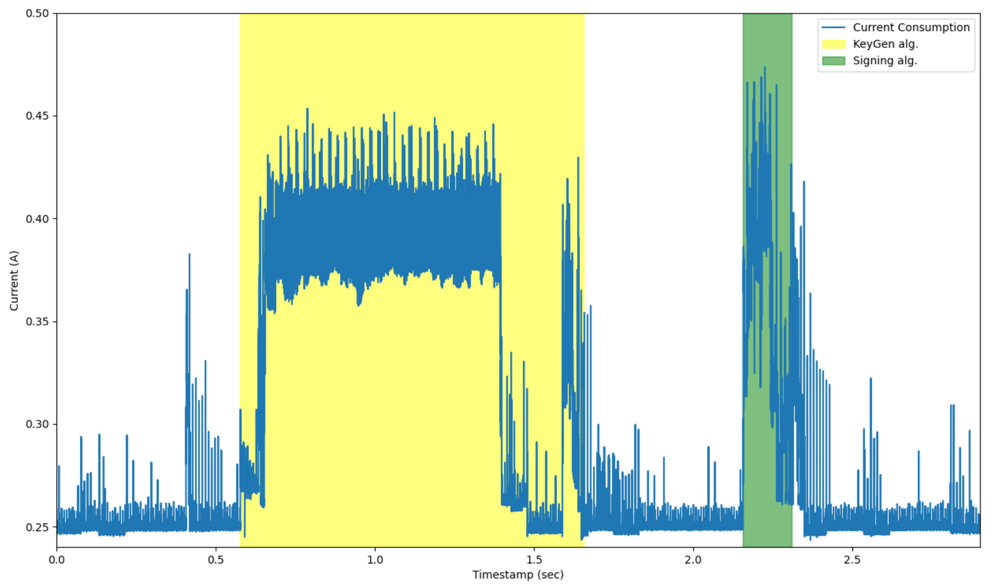


Figure 6.3: Current Consumption of DKLS

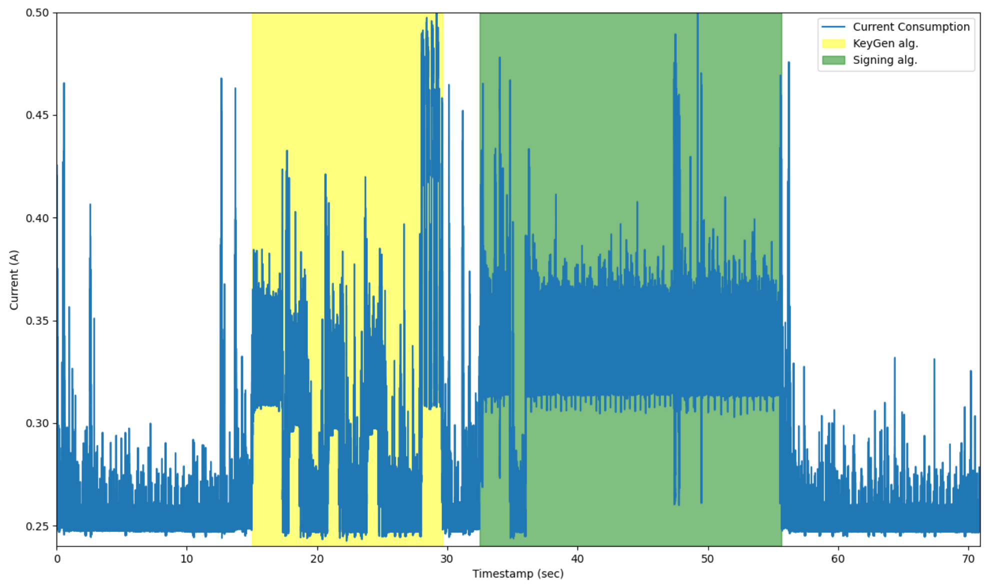


Figure 6.4: Current Consumption of GG20

is connected to the network via WiFi 5. Furthermore, to run the Hyperledger Fabric network, we used a laptop equipped with a 10-core Apple M1 processor, running at the clock speed of 2.0 – 3.2 GHz and 32 GB of RAM. Further, the FABRIC network runs on the same local network as the other devices.
Software: We used the same ZF FROST library as mentioned in subsection 6.1.1. Furthermore, we used Hyperledger Fabric 3.0 SDK. Within our Hyperledger Fabric network, we configured a single channel with a single ordering and two organizations, each owning one peer. We mainly used the default configuration for both peers and orderer including a Batch-timeout of 2 seconds among others. Further, each node runs within its dedicated docker container, running Linux Kernel 5.15.

6.2.2. Methodology

The second experiment was conducted using the four Raspberry Pi 2 devices and an additional Android smartphone, that executed our modified Fabric SDK equipped with the modified version of the FROST library with support for communication between 5 devices, and without the signature aggregator. One of the Raspberry Pi devices is used to initiate the process by creating a transaction and requesting to collaboratively sign the transaction. This device is responsible for submitting the signed transaction to the network. Similar to the previous experiment, we used various threshold configurations but with $n = 5$ devices, i.e., 5 is the maximal number of devices. Finally, the Fabric v3.0 blockchain was used with the support for ED25519 signature verification. In both experiments, the results are measured by taking the average of 10 trials while having different configurations.

6.2.3. Results

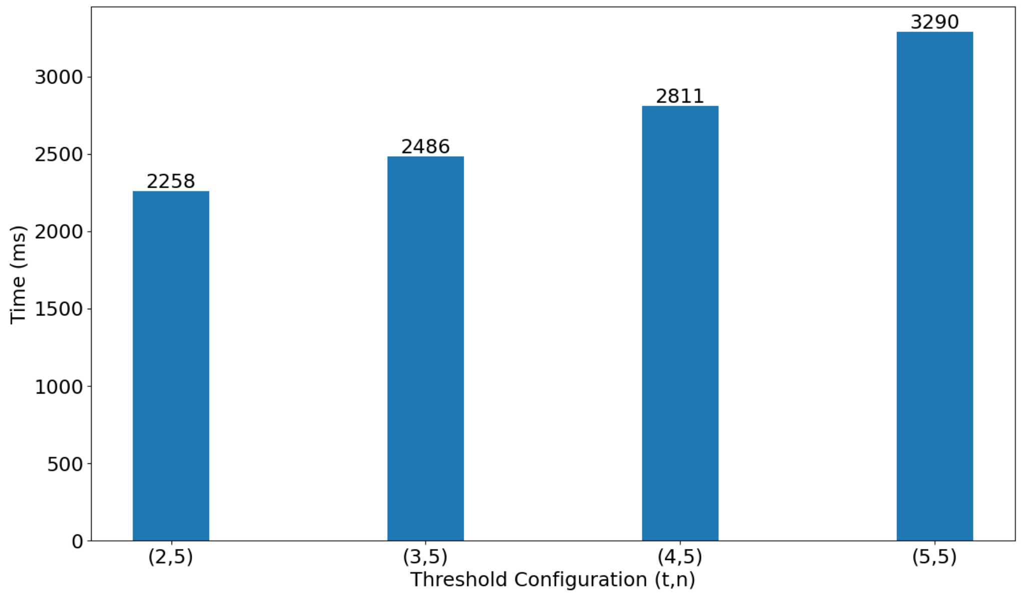


Figure 6.5: Transaction submission time using FROST within the Hyperledger Fabric

Figure 6.5 illustrates the time required to submit a single transaction to Hyperledger Fabric v3.0 using a threshold signature with FROST. The transaction contains a call to a smart contract function, which stores a unique integer value on the general ledger. In this experiment, we used the default batch timeout of 2 seconds and batch size of maximum 10 messages, for the orderer. A new block will be added to the blockchain if either the upper batch size limit is reached or the maximum number of transactions a single block can hold [22].

The results reveal a linear relationship between the number of signers and the time required to submit a transaction. Despite variations in the number of signers, the transaction submission times consistently adhere to the batch timeout of 2 seconds. When compared to findings from previous studies [63], our observed transaction submission times align within a similar range, affirming the effectiveness of FROST within the Hyperledger Fabric environment.

7 Discussions

Our study provides a detailed examination of threshold signature schemes, with a particular focus on the standout performance of the FROST algorithm, especially relevant in IoT contexts. We have shown that FROST's efficiency makes it highly suitable for use in mid-range IoT devices and smartphones, and its integration with Hyperledger Fabric v3.0 is practical. However, our research also points to limitations that require further exploration.

One such limitation is our experiments' exclusive reliance on LAN network conditions. Future research will need to explore the impact of varied network environments, e.g., WAN on FROST's performance to better understand its robustness and reliability in real-world settings.

Additionally, while FROST demonstrates efficient performance and effective communication rounds, its robustness requires further examination. The robustness is defined as the ability to guarantee a successful signing session with the presence of at least t honest signers, even if all remaining signers are malicious and try to prevent the honest participants from creating a valid signature. Notably, in the FROST protocol, the presence of t honest signers does not inherently guarantee the successful conclusion of a signing session. For instance, in a scenario with $t + 1$ signers where one signer is disruptive, FROST can identify the disruptive participant. However, the entire signing session will fail and require a restart.

To address this challenge, the ICE-FROST [29] and ROAST [56] protocols present a promising approach for improving the robustness of the FROST threshold signature scheme in the key and signature generation methods respectively. ICE-FROST enables robustness and support for static public keys in the key generation algorithm. Similarly, by satisfying the prerequisites of one preprocessing round and one online signing round, ROAST ensures that a signing session involving up to n signers will succeed and produce a valid signature as long as there are t honest signers are present, even if all remaining signers are disruptive.

As we look to the future, our research aims to not only refine the implementation of FROST within IoT and blockchain ecosystems but also to explore the integration of resilience mechanisms such as ICE-FROST and ROAST ensuring that threshold signature scheme can meet the demands of security, efficiency, and robustness in increasingly complex and varied application scenarios.

8 Conclusion

The integration of blockchain technologies into IoT applications introduces a set of challenges, notably in the management of digital signatures and keys. The practice of storing cryptographic keys on a single device creates a single point of failure. Moreover, the traditional multi-signature schemes not only compromise user privacy by revealing the identities of signers but also suffer from scalability issues. Furthermore, the application of conventional ECDSA-based threshold signatures is not efficient in the context of IoT devices, due to the high energy consumption and execution time.

To address these challenges, our study has successfully demonstrated the integration and advantages of the FROST signature protocol within IoT networks for signing transactions on the Hyperledger Fabric blockchain. Through our investigation, by comparing FROST with existing solutions, we have shown the benefits of adapting FROST within IoT blockchain environments, notably its reduced energy requirements and shorter execution times, making it particularly well-suited for mid-range IoT devices and smartphones.

The empirical findings of our integration of FROST with Hyperledger Fabric indicate the efficiency of FROST signatures, with group signatures executed in 58ms on Cortex-A7 CPU-based IoT architectures and a FROST-signed Fabric transaction commitment time of 3.2 seconds (with a 2-second batch timeout) in a 5 out of 5 device scenario. Furthermore, our proposal also facilitates the registration of a FROST group public key in Hyperledger Fabric v3.0 without compromising the group's private key.

These results demonstrate that FROST offers reasonable latency and energy consumption within a network of IoT and Hyperledger Fabric, leading us to conclude that IoT and Hyperledger Fabric blockchain technologies are now **"Completely FROST-ed"**. Our open-source implementation can be found on GitHub [1].

Finally, our contribution has been recognized by the research community, as the core research article of this master thesis was published in the IEEE International Conference on Blockchain and Cryptocurrency (IEEE/ICBC) 2024, one of the premier conferences in the field of blockchain.

[1] `https://github.com/mkhattat/Frost-Fabric-IoT`

References

[1] *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*. IEEE, 2017. ISBN: 978-1-5386-2244-5. URL: https://ieeexplore.ieee.org/xpl/conhome/7966454/proceeding.

[2] *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019. ISBN: 978-1-5386-6660-9. URL: https://ieeexplore.ieee.org/xpl/conhome/8826229/proceeding.

[3] Hal Abelson et al. "The risks of key recovery, key escrow, and trusted third-party encryption". In: *World Wide Web J.* 2 (1997), pp. 241–257.

[4] Omar Alfandi et al. "A survey on boosting IoT security and privacy through blockchain". In: *Cluster Computing* 24.1 (Mar. 2021), pp. 37–55. ISSN: 1573-7543. DOI: 10.1007/s10586-020-03137-8. URL: https://doi.org/10.1007/s10586-020-03137-8.

[5] Shikah J. Alsunaidi and Fahd A. Alhaidari. "A Survey of Consensus Algorithms for Blockchain Technology". In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. 2019, pp. 1–6. DOI: 10.1109/ICCISci.2019.8716424.

[6] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. Ed. by Rui Oliveira, Pascal Felber, and Y. Charlie Hu. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[7] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. Ed. by Rui Oliveira, Pascal Felber, and Y. Charlie Hu. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[8] Fredy Andres Aponte-Novoa et al. "The 51% Attack on Blockchains: A Mining Behavior Study". In: *IEEE Access* 9 (2021), pp. 140549–140564. DOI: 10.1109/ACCESS.2021.3119291. URL: https://doi.org/10.1109/ACCESS.2021.3119291.

[9] Michael Backes, Aniket Kate, and Arpita Patra. "Computational Verifiable Secret Sharing Revisited". In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 590–609. ISBN: 978-3-642-25384-3. DOI: 10.1007/978-3-642-25385-0_32. URL: https://doi.org/10.1007/978-3-642-25385-0%5C_32.

[10] Imran Bashir. *Mastering blockchain*. Packt Publishing Ltd, 2017.

[11] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. "Randomness Re-use in Multi-recipient Encryption Schemeas". In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 85–99. ISBN: 3-540-00324-X. DOI: `10.1007/3-540-36288-6\_7`. URL: `https://doi.org/10.1007/3-540-36288-6%5C_7`.

[12] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by Dorothy E. Denning et al. ACM, 1993, pp. 62–73. ISBN: 0-89791-629-8. DOI: `10.1145/168588.168596`. URL: `https://doi.org/10.1145/168588.168596`.

[13] Raynor de Best. *Blockchain technology use cases in organizations worldwide as of 2021*. `https://www.statista.com/statistics/878732/worldwide-use-cases-blockchain-technology/`. 2021.

[14] Aniruddha Bhattacharjya et al. "Security Challenges and Concerns of Internet of Things (IoT)". In: *Cyber-Physical Systems: Architecture, Security and Application*. Ed. by Song Guo and Deze Zeng. Cham: Springer International Publishing, 2019, pp. 153–185. ISBN: 978-3-319-92564-6. DOI: `10.1007/978-3-319-92564-6_7`. URL: `https://doi.org/10.1007/978-3-319-92564-6_7`.

[15] Aymen Boudguiga et al. "Towards Better Availability and Accountability for IoT Updates by Means of a Blockchain". In: *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 50–58. ISBN: 978-1-5386-2244-5. DOI: `10.1109/EUROSPW.2017.50`. URL: `https://doi.org/10.1109/EuroSPW.2017.50`.

[16] Ran Canetti et al. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti et al. ACM, 2020, pp. 1769–1787. ISBN: 978-1-4503-7089-9. DOI: `10.1145/3372297.3423367`. URL: `https://doi.org/10.1145/3372297.3423367`.

[17] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*. Ed. by Margo I. Seltzer and Paul J. Leach. USENIX Association, 1999, pp. 173–186. ISBN: 1-880446-39-1. URL: `https://dl.acm.org/citation.cfm?id=296824`.

[18] Ivan Damgård et al. "Fast threshold ECDSA with honest majority". In: *J. Comput. Secur.* 30.1 (2022), pp. 167–196. DOI: `10.3233/JCS-200112`. URL: `https://doi.org/10.3233/JCS-200112`.

[19] Omar Dib et al. "Consortium blockchains: Overview, applications and challenges". In: *Int. J. Adv. Telecommun* 11.1 (2018), pp. 51–64.

[20] Jack Doerner et al. "Threshold ECDSA from ECDSA Assumptions: The Multiparty Case". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1051–1066. ISBN: 978-1-5386-6660-9. DOI: `10.1109/SP.2019.00024`. URL: `https://doi.org/10.1109/SP.2019.00024`.

[21] Ali Dorri et al. "Blockchain for IoT security and privacy: The case study of a smart home". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2017, Kona, Big Island, HI, USA, March 13-17, 2017*. IEEE, 2017, pp. 618–623. ISBN: 978-1-5090-4338-5. DOI: `10.1109/PERCOMW.2017.7917634`. URL: `https://doi.org/10.1109/PERCOMW.2017.7917634`.

[22] Julian Dreyer, Marten Fischer, and Ralf Tönjes. "Performance analysis of hyperledger fabric 2.0 blockchain platform". In: *Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems*. CCIoT '20. Virtual Event, Japan: Association for Computing Machinery, 2020, pp. 32–38. ISBN: 9781450381314. DOI: `10.1145/3417310.3431398`. URL: `https://doi.org/10.1145/3417310.3431398`.

[23] Manu Drijvers et al. "On the Security of Two-Round Multi-Signatures". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1084–1101. ISBN: 978-1-5386-6660-9. DOI: `10.1109/SP.2019.00050`. URL: `https://doi.org/10.1109/SP.2019.00050`.

[24] Paul Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*. IEEE Computer Society, 1987, pp. 427–437. ISBN: 0-8186-0807-2. DOI: `10.1109/SFCS.1987.4`. URL: `https://doi.org/10.1109/SFCS.1987.4`.

[25] Wood G. "Ethereum: a secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), p. 1. URL: `https://cir.nii.ac.jp/crid/1370294643848527233`.

[26] 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, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 1179–1194. ISBN: 978-1-4503-5693-0. DOI: `10.1145/3243734.3243859`. URL: `https://doi.org/10.1145/3243734.3243859`.

[27] Rosario Gennaro and Steven Goldfeder. "One Round Threshold ECDSA with Identifiable Abort". In: *IACR Cryptol. ePrint Arch.* (2020), p. 540. URL: `https://eprint.iacr.org/2020/540`.

[28] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208. DOI: `10.1137/0218012`. URL: `https://doi.org/10.1137/0218012`.

[29] Alonso González et al. "Identifiable Cheating Entity Flexible Round-Optimized Schnorr Threshold (ICE FROST) Signature Protocol". In: *IACR Cryptol. ePrint Arch.* (2021), p. 1658. URL: `https://eprint.iacr.org/2021/1658`.

[30] Harry Halpin and Marta Piekarska. "Introduction to Security and Privacy on the Blockchain". In: *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 1–3. ISBN: 978-1-5386-2244-5. DOI: `10.1109/EUROSPW.2017.43`. URL: `https://doi.org/10.1109/EuroSPW.2017.43`.

[31] K. ITAKURA. "A public-key cryptosystem suitable for digital multisignature". In: *NEC Research and Development* 71 (1983), pp. 1–8. URL: `https://cir.nii.ac.jp/crid/1572543025896144128`.

[32] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *Int. J. Inf. Sec.* 1.1 (2001), pp. 36–63. DOI: `10.1007/S102070100002`. URL: `https://doi.org/10.1007/s102070100002`.

[33] Sunny King. "Primecoin: Cryptocurrency with prime number proof-of-work". In: *July 7th* 1.6 (2013).

[34] Sunny King and Scott Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake". In: *self-published paper, August* 19.1 (2012).

[35] Chelsea Komlo and Ian Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn. Vol. 12804. Lecture Notes in Computer Science. Springer, 2020, pp. 34–65. ISBN: 978-3-030-81651-3. DOI: 10.1007/978-3-030-81652-0_2. URL: https://doi.org/10.1007/978-3-030-81652-0%5C_2.

[36] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176.

[37] Dongcheng Li, W. Eric Wong, and Jincui Guo. "A Survey on Blockchain for Enterprise Using Hyperledger Fabric and Composer". In: *6th International Conference on Dependable Systems and Their Applications, DSA 2019, Harbin, China, January 3-6, 2020*. IEEE, 2019, pp. 71–80. ISBN: 978-1-7281-6057-3. DOI: 10.1109/DSA.2019.00017. URL: https://doi.org/10.1109/DSA.2019.00017.

[38] David Lie et al., eds. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018. ISBN: 978-1-4503-5693-0. URL: http://dl.acm.org/citation.cfm?id=3243734.

[39] Yehuda Lindell and Ariel Nof. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 1837–1854. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243788. URL: https://doi.org/10.1145/3243734.3243788.

[40] Eliza Mik. "Smart contracts: terminology, technical limitations and real world complexity". In: *Law, Innovation and Technology* 9.2 (2017), pp. 269–300. DOI: 10.1080/17579961.2017.1378468. eprint: https://doi.org/10.1080/17579961.2017.1378468. URL: https://doi.org/10.1080/17579961.2017.1378468.

[41] Natalia G. Miloslavskaya and Alexander I. Tolstoy. "Internet of Things: information security challenges and solutions". In: *Clust. Comput.* 22.1 (2019), pp. 103–119. DOI: 10.1007/S10586-018-2823-6. URL: https://doi.org/10.1007/s10586-018-2823-6.

[42] Daniel Minoli and Benedict Occhiogrosso. "Blockchain mechanisms for IoT security". In: *Internet Things* 1-2 (2018), pp. 1–13. DOI: 10.1016/J.IOT.2018.05.002. URL: https://doi.org/10.1016/j.iot.2018.05.002.

[43] Satoshi Nakamoto et al. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).

[44] Magnus Nyström and Burt Kaliski. "PKCS #10: Certification Request Syntax Specification Version 1.7". In: *RFC* 2986 (2000), pp. 1–14. DOI: 10.17487/RFC2986. URL: https://doi.org/10.17487/RFC2986.

[45] Rui Oliveira, Pascal Felber, and Y. Charlie Hu, eds. *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3190508.

[46] Aafaf Ouaddah, Anas Abou El Kalam, and Abdellah Ait Ouahman. "FairAccess: a new Blockchain-based access control framework for the Internet of Things". In: *Secur. Commun. Networks* 9.18 (2016), pp. 5943–5964. DOI: 10.1002/SEC.1748. URL: https://doi.org/10.1002/sec.1748.

[47] Kazim Rifat Ozyilmaz and Arda Yurdakul. “Designing a Blockchain-Based IoT With Ethereum, Swarm, and LoRa: The Software Solution to Create High Availability With Minimal Security Risks”. In: *IEEE Consumer Electron. Mag.* 8.2 (2019), pp. 28–34. DOI: `10.1109/MCE.2018.2880806`. URL: `https://doi.org/10.1109/MCE.2018.2880806`.

[48] Torben P. Pedersen. “A Threshold Cryptosystem without a Trusted Party (Extended Abstract)”. In: *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Springer, 1991, pp. 522–526. ISBN: 3-540-54620-0. DOI: `10.1007/3-540-46416-6\_47`. URL: `https://doi.org/10.1007/3-540-46416-6%5C_47`.

[49] Wouter Penard and Tim van Werkhoven. “On the secure hash algorithm family”. In: *Cryptography in context* (2008), pp. 1–18.

[50] David Pointcheval and Jacques Stern. “Security Arguments for Digital Signatures and Blind Signatures”. In: *J. Cryptol.* 13.3 (2000), pp. 361–396. DOI: `10.1007/S001450010003`. URL: `https://doi.org/10.1007/s001450010003`.

[51] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, 1996, pp. 387–398. ISBN: 3-540-61186-X. DOI: `10.1007/3-540-68339-9\_33`. URL: `https://doi.org/10.1007/3-540-68339-9%5C_33`.

[52] Rajat Rathi et al. “Security Challenges Controls in Cyber Physical System”. In: *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*. 2020, pp. 242–247. DOI: `10.1109/CSNT48778.2020.9115778`.

[53] Chenshan Ren et al. “Distributed Online Optimization of Fog Computing for Internet of Things Under Finite Device Buffers”. In: *IEEE Internet Things J.* 7.6 (2020), pp. 5434–5448. DOI: `10.1109/JIOT.2020.2979353`. URL: `https://doi.org/10.1109/JIOT.2020.2979353`.

[54] Sara Ricci et al. “Threshold Signature for Privacy-Preserving Blockchain”. In: *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum*. Ed. by Andrea Marrella et al. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2022, pp. 100–115. ISBN: 978-3-031-16168-1. DOI: `10.1007/978-3-031-16168-1_7`.

[55] Karen Rose, Scott Eldridge, and Lyman Chapin. “The internet of things: An overview”. In: *The internet society (ISOC)* 80 (2015), pp. 1–50.

[56] Tim Ruffing et al. “ROAST: Robust Asynchronous Schnorr Threshold Signatures”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by Heng Yin et al. ACM, 2022, pp. 2551–2564. ISBN: 978-1-4503-9450-5. DOI: `10.1145/3548606.3560583`. URL: `https://doi.org/10.1145/3548606.3560583`.

[57] Mehrdad Salimitari and Mainak Chatterjee. “An Overview of Blockchain and Consensus Protocols for IoT Networks”. In: *CoRR* abs/1809.05613 (2018). arXiv: `1809.05613`. URL: `http://arxiv.org/abs/1809.05613`.

[58] Shivam Saxena, Bharat Bhushan, and Mohd Abdul Ahad. “Blockchain based solutions to secure IoT: Background, integration trends and a way forward”. In: *J. Netw. Comput. Appl.* 181 (2021), p. 103050. DOI: `10.1016/J.JNCA.2021.103050`. URL: `https://doi.org/10.1016/j.jnca.2021.103050`.

[59] Claus P Schnorr. *Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system*. US Patent 4,995,082. Feb. 1991.

[60] Claus Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptol.* 4.3 (1991), pp. 161–174. DOI: `10.1007/BF00196725`. URL: `https://doi.org/10.1007/BF00196725`.

[61] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 239–252. ISBN: 3-540-97317-6. DOI: `10.1007/0-387-34805-0\_22`. URL: `https://doi.org/10.1007/0-387-34805-0%5C_22`.

[62] Sureshkumar Selvaraj and Suresh Sundaravaradhan. "Challenges and opportunities in IoT healthcare systems: a systematic review". In: *SN Applied Sciences* 2.1 (Dec. 2019), p. 139. ISSN: 2523-3971. DOI: `10.1007/s42452-019-1925-y`. URL: `https://doi.org/10.1007/s42452-019-1925-y`.

[63] Salma Shalaby et al. "Performance Evaluation of Hyperledger Fabric". In: *IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020, Doha, Qatar, February 2-5, 2020*. IEEE, 2020, pp. 608–613. ISBN: 978-1-7281-4821-2. DOI: `10.1109/ICIOT48696.2020.9089614`. URL: `https://doi.org/10.1109/ICIoT48696.2020.9089614`.

[64] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: `10.1145/359168.359176`. URL: `https://doi.org/10.1145/359168.359176`.

[65] Sabrina Sicari et al. "Toward Data Governance in the Internet of Things". In: *New Advances in the Internet of Things*. Ed. by Ronald R. Yager and Jordán Pascual Espada. Cham: Springer International Publishing, 2018, pp. 59–74. ISBN: 978-3-319-58190-3. DOI: `10.1007/978-3-319-58190-3_4`. URL: `https://doi.org/10.1007/978-3-319-58190-3_4`.

[66] S.R. Subramanya and B.K. Yi. "Digital signatures". In: *IEEE Potentials* 25.2 (2006), pp. 5–8. DOI: `10.1109/MP.2006.1649003`.

[67] Nick Szabo. "The idea of smart contracts". In: *Nick Szabo's papers and concise tutorials* 6.1 (1996), p. 199.

[68] Don Tapscott and Alex Tapscott. "The impact of the blockchain goes beyond financial services". In: *Harvard business review* 10.7 (2016).

[69] Qi Xia et al. "MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain". In: *IEEE Access* 5 (2017), pp. 14757–14767. DOI: `10.1109/ACCESS.2017.2730843`. URL: `https://doi.org/10.1109/ACCESS.2017.2730843`.

[70] Rebecca Yang et al. "Public and private blockchain in construction business process and information integration". In: *Automation in Construction* 118 (2020), p. 103276. ISSN: 0926-5805. DOI: `https://doi.org/10.1016/j.autcon.2020.103276`. URL: `https://www.sciencedirect.com/science/article/pii/S0926580520301886`.

[71] Keping Yu et al. "A Blockchain-Based Shamir's Threshold Cryptography Scheme for Data Protection in Industrial Internet of Things Settings". In: *IEEE Internet Things J.* 9.11 (2022), pp. 8154–8167. DOI: `10.1109/JIOT.2021.3125190`. URL: `https://doi.org/10.1109/JIOT.2021.3125190`.

[72] Vlad Zamfir. *Introducing Casper "the Friendly Ghost"*. `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost`. 2015.

[73] Yu Zhang and Jiangtao Wen. "The IoT electric business model: Using blockchain technology for the internet of things". In: *Peer-to-Peer Netw. Appl.* 10.4 (2017), pp. 983–994. DOI: 10.1007/S12083-016-0456-1. URL: https://doi.org/10.1007/s12083-016-0456-1.

[74] Zibin Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, June 25-30, 2017*. Ed. by George Karypis and Jia Zhang. IEEE Computer Society, 2017, pp. 557–564. ISBN: 978-1-5386-1996-4. DOI: 10.1109/BIGDATACONGRESS.2017.85. URL: https://doi.org/10.1109/BigDataCongress.2017.85.

[75] Zibin Zheng et al. "An overview on smart contracts: Challenges, advances and platforms". In: *Future Gener. Comput. Syst.* 105 (2020), pp. 475–491. DOI: 10.1016/J.FUTURE.2019.12.019. URL: https://doi.org/10.1016/j.future.2019.12.019.

[76] Zibin Zheng et al. "Blockchain challenges and opportunities: a survey". In: *Int. J. Web Grid Serv.* 14.4 (2018), pp. 352–375. DOI: 10.1504/IJWGS.2018.10016848. URL: https://doi.org/10.1504/IJWGS.2018.10016848.