Completely FROST-ed: IoT issued FROST signature for Hyperledger Fabric blockchain

***Abstract*—Today, there is an increasing need for an efficient threshold signatures that enforce the protection of identities and multi device-based authentication when interacting with a blockchain technology. This study presents a comprehensive analysis of threshold signatures, establishing FROST as the most efficient scheme in terms of performance. We uniquely demonstrate FROST's adaptability with empirical results, showcasing its feasibility on middle-range IoT devices and smartphones. In addition we propose an implementation, with a primary goal to enable IoT devices interaction with Hyperledger Fabric v3.0 using FROST for transaction signing. An IoT network of 5 devices can perform a signature and commit to the blockchain ledger in 3.2 seconds, when network latency is optimal.**

***Index Terms*—IoT, FROST, Blockchain, Privacy**

I. Introduction

According to the statistics [1] about blockchain-based use cases applied in organizations worldwide, more than 40% of the companies use blockchain for digital currencies and almost the same amount uses blockchain for asset tracking and management and for digital identification. In recent years, the increased interest in digital currencies within the industry captured the attention of academic researches, leading to new investigation to improve threshold signatures, and replace the multi-signature schemes which are built in some of the blockchain technologies, such as Bitcoin [2]. The main issue with multi-signature schemes is that as many signatures are generated as signers, which impacts the verification time and storage place needed in the blockchain.

Unlike multi-signature schemes, the threshold signatures, also known as "t-of-n" signature generate a single signature that is signed at least by t of the n possible signers. Another advantage of these schemes is that the signature is verified using a single threshold public key, also known as group public key. As a result, the blockchain does not have to store the public keys of n signers, but only the group public key.

Many threshold-based signature schemes [3]–[7] were proposed in the recent years with the principal goal of signing blockchain transactions. These schemes are all based on Elliptic Curve Digital Signature Algorithm (ECDSA) signature that implies their direct compatibility with most blockchains, as they also employ ECDSA signature verification. However, the ECDSA-based threshold signatures involve a high execution time due to the high number of communication rounds that signers must perform among themselves and the difficulty of the ECDSA signatures. To address this issue new Shnorr signature based schemes were proposed [8]–[12]. Recently, the FROST signature [8] has received substantial attention in research, in such a manner that [11], [12] signatures now employs FROST as a core component. However, most of today's blockchain technology uses ECDSA signature for blockchain transaction verification, thus the in-built Schnorr signature verification is not feasible. Only Bitcoin and Bitcoin-based blockchains e.g., Liquid and RSK adopt this signature. Since it is challenging to modify and migrate already implemented public blockchains' structure to adopt Schmorr signature, we might expect that these signatures will be more popular in permissioned blockchains due to their more modular architecture.

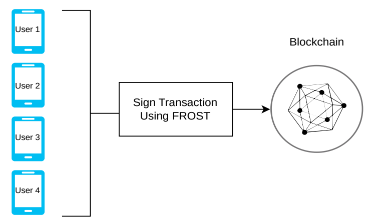
The adaption of threshold signatures in permissioned blockchain-based financial systems and ecosystems offers clear benefits, since at least t signers agreement is needed. The agreement results in a single signature that therefore allows a more trustworthy decision making in cryptocurrency transfers and smart contract processes execution. According to the previously mentioned statistics the integration of threshold signatures into blockchain technology is also advantageous for reinforcing the digital identification. Hence, blockchain-based ecosystem could have benefit by using IoT devices such as smartphones, and smartwatches for a more secure digital identification and agreement on cryptocurrency transfers and smart contract processes execution.

In this work, we also provide a rich state of the art about today's threshold signatures, highlighting the main differences between the ECDSA- and Schnorr-based schemes, and showing performance and characteristic differences. Hyperledger Fabric [13] is one of the most popular permissioned blockchain, mainly used in enterprise and ecosystem level use cases, and according to its road map of development, the version 3.0 will enable the application of ED25519 signature by default. This signature is a Schnorr variant.

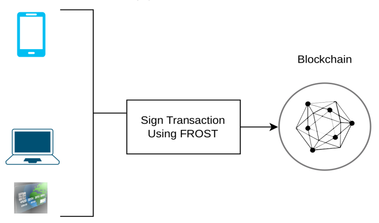
Since FROST signature received a significant attention in research and thanks to its promising performance and characteristics (will be further discussed in Sect. III), we believe it is now time to evaluate on the adaptability of an IoT-executed FROST signature within the Hyperledger Fabric blockchain network.

A. Motivation and contribution

As a motivation, in the following we present two real-life use case scenarios in which an IoT-Blockchain system will be beneficial thanks to integrating it with FROST threshold signature algorithm. Figure 1 illustrates these two use cases.
Multi-User In this variation, the primary focus is on protecting the identities of users participating in transactions. All users collectively own a common wallet on the blockchain network



(a) Multi-User



(b) Multi-Device

Fig. 1: Frost threshold signature use case

and the associated key is distributed among them [9]. The collaborative signing process will result in a single signature which is verifiable by the wallet's public key, thus concealing the identities of individual users. This scenario is particularly interesting in joint asset management and shared financial transactions.

Multi-Device The multi-device use case prioritizes the security of the blockchain wallet itself. In this scenario a user owns a wallet and the associated secret key is distributed among multiple devices that belong to the user [9]. In this approach, we can mitigate the risk of a single point of failure, as the collaboration of multiple devices is required to submit a transaction. The contributions of our work are as follows:

- Comparison of recently proposed threshold signature schemes
- Deployment of APIs on the top of Hyperledger Fabric SDK to enable the transaction signing with FROST signatures algorithm and the registration for group of devices to the Hyperledger Fabric network
- Deployment of an IoT-enabled Android-based application combining FROST, and Hyperledger Fabric SDK
- Testing FROST signature verification on Hyperledger Fabric v3.0

The structure of our article is as following: Sect. II describes the preliminaries of FROST signature, blockchain technology, and Hyperledger Fabric blockchain. Sect. III provides a rich state of the art on threshold signatures. Our proposed protocol is presented in Sect. IV. The experiments and results of the proposed solution are showcased in Sect. V. A discussion of our work is provided in Sect. VI. Finally, Sect VII concludes our study.

II. Preliminaries

In this section we describe in nutshell, the FROST signature, blockchain technology and Hyperledger Fabric blockchian.

A. *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 [8]. The core characteristic of FROST is its trade-off between network efficiency and robustness. 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 [8].

B. *Blockchain*

In a nutshell, blockchain is distributed ledger technology that allows members connected in a peer-to-peer topology to share an identical database in an immutable and transparent manner [14]. Furthermore, a new data entity is added to the distributed database according to a common agreement of the participants, known as the consensus rule [15]. Since blockchain 2.0 the implementation of smart contracts is also possible and it allows the automatic, transparent, and trustworthy execution of business logic described in format of digital code [16].

Blockchain technology is divided into two classes: public (permissionless) and private (permissioned) blockchains. In the class of public blockchains e.g., Ethereum [17] and Bitcoin [2] everybody cab join and have access to read and write data to the ledger. In private blockchains e.g., Hyperledger [13] Fabric the registration of a new member is issued in accordance of authorities related to organizations participating in the blockchain network. Moreover, read-write accesses depend on the policy made by the organizations. Hyperledger Fabric is one of the most popular permissioned blockchain used in enterprise and organizational ecosystem applications, thanks to its modularity and compatibility with employable smart contracts in different programming languages.

C. *Hyperledger Fabric*

Hyperledger Fabric uses Certificate Authorities which are systems responsible for confirming the identities of entities and linking them to cryptographic keys through the issuance of electronic documents known as digital certificates. In Hyperledger Fabric these systems are called "Fabric CAs" and they issue signed digital certificates following the X.509 standard. The Fabric CA provides two essential features: *registration of identities*, *issuance of enrollment certificates*. New identity registration to the Fabric network is issued as follows: first, the administrator initiate a registration request of a new user to the Fabric CA. The request includes information about the user, such as its affiliation, country, etc. Next, the Fabric CA generates the public-private key pair and the Certificate Signing Request (CSR) which is used to generate the X.509 digital certificate signed by the Fabric CA. Afterwards, the user can be enrolled to the Fabric CA with providing its enrollment ID and password. Finally the Fabric CA provides the X.509 certificate and the generated secret key to the user. It must be noted, that the self-signed X.509 certificate is essential for creating valid blockchain transactions issued in Hyperledger Fabric client applications.

III. Related Work

Only recently that extensive research exploring this concept of threshold signatures applied in the context of blockchain technologies has begun. In the followings, we classify the threshold signatures found in related work with taking into account their features, such as the performance, highlighting the communication rounds required. The signature algorithm that the schemes are based on. We also examine the majority assumption setting, which can be either honest or dishonest. The honest majority assumption assumes that at least half of the participants behave honestly and follow the protocol i.e the protocol supports a threshold settings of $t \leq (n-1)/2$. In contrast, in the dishonest majority setting, only one participant needs to be honest which allows a protocol with n participant to support a threshold configuration of $(n-1, n)$.

The identification of aborts, meaning that a signatory member stopped or does not follow the protocol. Online non-interactivity implies that each signatory generates their individual partial signature after having seen the message, without requiring interaction with any other signatory. The following signature algorithms have been implemented primary for the purpose of application in blockchain and cryptocurrency context, and one of them has been used in IoT-blockchain-based system.

Initially, Goldfeder et al. [4] introduced threshold signatures for Bitcoin. They argue that the current blockchain technologies, especially Bitcoin, lack support for the sophisticated internal control systems deployed by modern businesses. They suggested that a threshold signature scheme could be a solution to bridge this gap.

Lindel et al. [3] presented one of the earliest efficient protocols to provide threshold ECDSA. This protocol replaces Pailier additive homomorphic encryption with the in-the-exponent version of ElGamal encryption. It requires five rounds of communication for the key generation and 8 for signing.

GG20 [5] improved the efficiency of GG18 [18] by reducing the number of communication rounds and new mechanisms to identify and deal with misbehaving parties who deviate from the protocol. Key generation involves 4 communication rounds while signing requires 7 rounds. GG20 assumes a majority of parties may be dishonest and allows for an offline, pre-processing phase for signing.

Damgard et al. [6] introduced a schema with the assumption of 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.

Furthermore, [7] is the only one 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 a majority of participants to be dishonest.

In addition, [19] introduced a threshold scheme requiring 3 rounds of communication for the key generation, an additional 3 rounds for the key refreshing algorithm, and 4 rounds for signing (including 3 offline rounds). It presents 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 efficiency bottleneck [18].

In [9], 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 communications. The signing algorithm is divided into two phases and requires three rounds of communications between the main device and the secondary devices. In the last round the main device aggregates all signature fragments and produces a Schnorr signature.

The schemes proposed by Ruffing *et al.* and Alonso *et al.* [11], [12] used FROST as core component in order to make it more robust semi-interactive signing protocol. In these protocols, the robustness ends up in more significant number of communication rounds. Table I summarizes the discussed threshold signatures found in the related work, 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 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, though this may require restarting the protocol [8]. These qualities make FROST a strong candidate for the efficient utilization of Internet of Things (IoT) devices in blockchain

TABLE I: Summary of threshold signatures and their characteristics

Protocol	Core algorithm	Performance	Majority	Identifiable Aborts	Online Non-Interactivity
Lindell (2018) [3]	ECDSA	Key Gen: 5 rounds Signing: 8 rounds	Dishonest	✗	✗
GG20 (2020) [5]	ECDSA	Key Gen: 4 rounds Signing: 7 rounds	Dishonest	✓	✓
Damgard (2022) [6]	ECDSA	Key Gen: 3 rounds Signing: 4 rounds	Honest $t \leq (n-1)/2$	✗	✓
DKLS (2019) [7]	ECDSA	Key Gen: 8 rounds Signing: $6 + \log(t)$	Dishonest	✗	✓
CMP (2020) [19]	ECDSA	Key Gen: 3 rounds Key Refresh: 3 rounds Signing: 4 rounds	Dishonest	✓	✓
RDCC (2022) [9]	Schnorr	Key Gen: 2 rounds Signing: 3 rounds	Dishonest	✗	✗
ROAST (2022) [11]	Schnorr	Key Gen: N/A Signing: ¿ 3 rounds	Dishonest	✓	✗
ICE-Frost (2023) [12]	Schnorr	Key Gen: 2 rounds Complaint Management: 2 rounds Signing: 2 rounds	Dishonest	✓	✓
FROST (2020) [8]	Schnorr	Key Gen: 2 rounds Signing: 2 rounds	Dishonest	✓	✓

applications, which is central to our research focus. It is worth noting that the protocols [11], [12] utilize FROST as a main component, with reasonable overhead in terms of communication rounds. Given that one of our primary objective is to develop an efficient blockchain-based IoT-based threshold signatures protocol we select FROST signature as core element of our proposed protocol. Furthermore, we assume that as our proposed solution is designed to work with FROST the aforementioned, FROST-based protocol will also be compatible with our solution, but a reasonably higher latency might occur.

IV. Proposed Protocol

In this section, we specify the setup in which our IoT-enabled FROST signing operates 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 Hyperledger Fabric network, particularly when the registration and enrollment procedure involves Fabric Certificate Authority. Finally, we demonstrate the entire transaction submission process, covering its generation, signing by the devices using FROST and its submission to the blockchain network.

A. Setup of the network architecture

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

- There exists 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.

B. Registering and Enrolling with a Certificate Authority

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

In sect. II 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 it is not feasible. In FROST signature protocol, each signer possesses a share of the group private key, which is collectively generated by each participant of the signatory group. Additional details about FROST key generation algorithm can be found in [8]. 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.

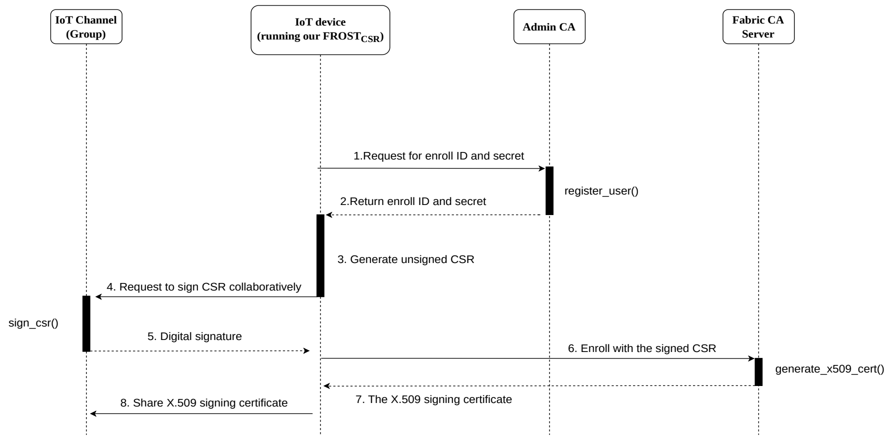

IoT Channel (Group)
IoT device (running our FROST$_{CSR}$)
Admin CA
Fabric CA Server
1.Request for enroll ID and secret
register_user()
2.Return enroll ID and secret
3. Generate unsigned CSR
4. Request to sign CSR collaboratively
sign_csr()
5. Digital signature
6. Enroll with the signed CSR
generate_x509_cert()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

Fig. 2: The registration and enrollment flow

It is important to note that with this approach the private key will never be shared to the admin CA.

In case of threshold signature, the main challenge is to send 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 [20]: *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 2, the registration and enrollment process begins with a user delegated by group of users, 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, group's public key, and the signature algorithm identifier i.e, ED25519. The 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 sign certificate to the other members. Note that possession of a sign 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.

C. Transaction generation and signing API

In the context of our proposed protocol depicted in Figure 2, 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, which provides tools for interacting with the blockchain network, includes an interface called Gateway to communicate with the blockchain network. In this interface, the clients can submit a custom signing message which in our protocol will be 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 3 shows the transaction flow.

In conclusion, we can affirm that the registration and enrollment is enabled with our proposed 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.

D. 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

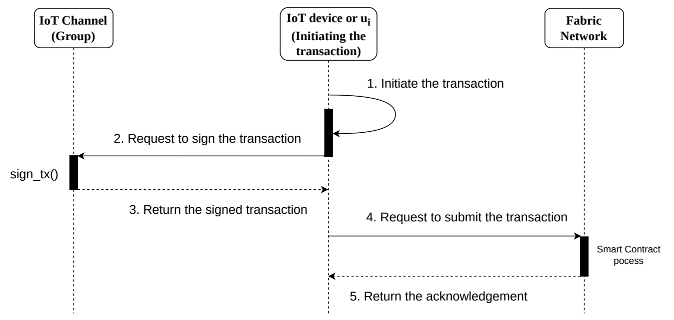

IoT Channel (Group)
IoT device or u_i (Initiating the transaction)
Fabric Network
1. Initiate the transaction
2. Request to sign the transaction
sign_tx()
3. Return the signed transaction
4. Request to submit the transaction
Smart Contract pocess
5. Return the acknowledgement

Fig. 3: The transaction flow

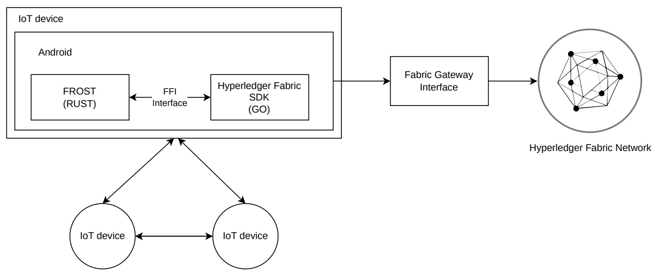


Fig. 4: The Architecture Overview

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 [8]. 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 message to a specific device within the group. Given that all devices operate on local network environment, we omitted the Transport Layer Security (TLS) protocol in this communication channel.

Furthermore, we implemented $\text{FROST}_{\text{CSR}}$ in the Go programming language to generate unsigned CSR which later will be sent to other participant 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 the user the use of 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 exchanging data between the two languages. In our implementation we used raw pointers to byte arrays to transmit 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 Android platform. This approach minimized overhead and enabled us to efficiently run and test our API implementation. Figure 4 shows an overview of all components used in the implementation. Note that all IoT devices follow the same structure. Further, in Figure 4 we used Android as an example of an IoT device.

V. Experiment

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

A. 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. 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 has been compiled and executed natively on the respective devices.

Network Environment: To ensure optimal network conditions for our experiment, we conducted our experiment on a 1Gbps LAN network. The network configuration is as follows: The four raspberry Pi devices are connected to the network via an Ethernet cable. The Android device is connected to the network via WiFi.

FROST Library: We used ZF FROST library version 0.7.0 which is implemented in Rust and provides the core implementation of the FROST key generation and signing algorithm. 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 key generation/signing and signing aggregator removal.

Hyperledger Fabric Network Configuration: 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. The entire network operated on a laptop equipped with a 10-core Apple M1 processor, running at clock speed of $2.0 - 3.2$ GHz and 32 GB of RAM. Further, the FABRIC network runs on the same local network as other devices.

B. Methodology

We conducted two experiments. In the first experiment, we focused on assessing the performance of FROST when executed on resource-constrained IoT devices. We conducted the experiment using four Raspberry Pi devices. Various threshold configurations, including 2-of-4, 3-of-4, and 4-of-4, were used to evaluate the performance of key generation and signing algorithms. To capture timing data, we measured the execution time of the key generation and the signing algorithms in milliseconds.

In the second experiment, we integrated FROST with the Hyperledger Fabric blockchain network to evaluate its performance in the context of transaction submissions by IoT devices. This experiment was conducted using the four Raspberry Pi 2 devices and the one Android smartphone, that executed our modified Fabric SDK with the equipped with the modified version of 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 to submit the signed transaction to the network. Similar to previous experiment, we used various threshold configuration but with $n = 5$ devices, i.e., 5 is the maximal number of devices. Finally, Fabric v3.0 blockchain was used with the support for ED25519 signature verification. In both experiments, the results are measured by taking average of 10 trials while having different configurations.

C. Results

Figure 5 illustrates the results for the first experiment, measuring the performance of both key generation and signing algorithm across varying threshold configurations. The key generation algorithm shows a relatively consistent performance across various threshold configurations and the number of devices, with execution time of around 22ms. On the other hand, the signing algorithm demonstrated a linear relationship with the number of devices participating in the signing protocol, with the execution time increasing proportionally to the number of devices. These results demonstrate the feasibility of using FROST in IoT devices.

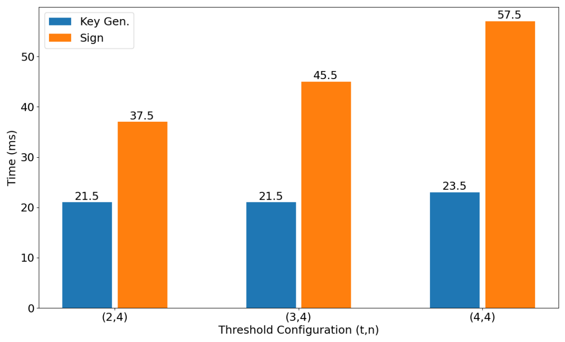


Fig. 5: Running times of FROST

Figure 6 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 transaction a single block can hold [21]. The results show a linear relation between the number of signers and the amount of time that it takes to submit a transaction. Considering the batch timeout of 2 seconds and in comparison with results from other studies [22], our transaction submission times fall within a similar range.

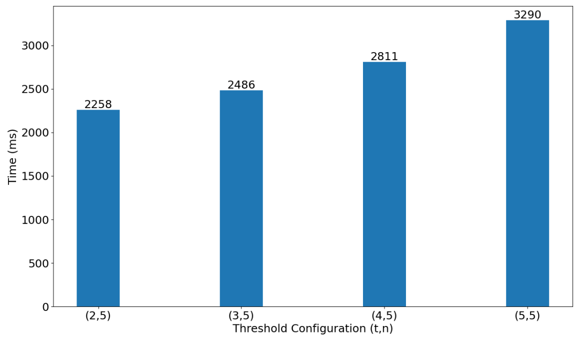


Fig. 6: Transaction submission time using FROST within the Hyperledger Fabric

VI. Discussions

In this study, we first present a comprehensive state of the art on threshold signatures, demonstrating that FORST is the most efficient scheme among the related work in terms of performance. Thanks to its efficiency FROST is an ideal choice for being adopted with IoT. However, our work is the first study showing its adaptability with empirical results. Our experiments highlight that the execution time of FORST in middle range IoT devices and smartphones is completely feasible with a reasonable execution time. In addition, our approach and proposed APIs enable the usage of FROST for Hyperledger Fabric v3.0. Since the performance of FROST in a combined network consisting of IoT and Hyperledger Fabric blockchain shows a reasonable latency, we can conclude that IoT and Hyperledger Fabric blockchain technologies are now "**Completely FROST-ed**".

However there are also limitations in our work that we have to discuss. While the FROST algorithm demonstrates efficient performance and effective communication rounds, it should be noted that it lacks robustness. Robustness can be defined as the guarantee that a signing session (involving up to n signers) will succeed and produce a valid signature if t honest signers are present in the signing session [11]. In case where k participants are present in the signing session where $t \leq k \leq n$, and one of the participants misbehaves by not adhering to the protocol or providing a malformed share, the FROST protocol detects this misbehavior and requires a restart of the signing process, even if t honest signers are present [8].

A promising solution to enhance the robustness of threshold signature schemes is presented by ROAST [11]. Under some requirements ROAST can be applied to a threshold scheme. To meet the requirements of ROAST a threshold scheme must have one prepossessing round and one actual signing round. Further, it must support identifiable aborts and be unforgable under concurrent signing sessions.

VII. Conclusion

In this study, we investigate the integration of the FROST signature protocol within the context an IoT-network that signs valid Hyperledger Fabric blockchain transactions. The adaption of FROST in IoT and permissioned blockchain technology can be particularly beneficial when user wants to identify her/himself with multiple devices i.e., the multiple devices has to sign a given transaction. Our proposed protocol enables the registration of FROST group public key in Hyperledger Fabric v3.0 via the usage of Fabric Certificate Authority (CA), but without sharing the group private key with the Fabric CA. In addition, with the modified Fabric SDK that part of our work, the IoT devices now enabled to execute FROST signature on Fabric blockchain transactions. Our work showcases the potential of FROST for secure and efficient threshold signature in IoT-blockchain applications, as demonstrated in our experiment, a 4 out of 4 group signature can be performed in around 58ms when using Cortex-A7 CPU based IoT architectures. Moreover, a FROST signed Fabric transaction is committed in a reasonable 3.2 seconds when 5 out of 5 devices signed the transaction. Finally, we can confirm that the future release of Hyperledger Fabric namely the version 3.0 is now able to accept FROST-signed transactions, thanks to our modified Fabric SDK. Moreover, our proposed registration protocol, enable Fabric version 3.0 the registration of group public keys instead of individual user public keys. Thanks to this modification the size of PKI can be reduced. Our implementation will be available on GitHub after acceptance of the article.

References

[1] R. de Best, "Blockchain technology use cases in organizations worldwide as of 2021 ," https://www.statista.com/statistics/878732/worldwide-use-cases-blockchain-technology/, 2021.

[2] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] Y. Lindell and A. 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*, 2018, pp. 1837–1854.

[4] S. Goldfeder, J. Bonneau, J. Kroll, and E. Felten, "Securing bitcoin wallets via threshold signatures," 2014.

[5] R. Gennaro and S. Goldfeder, "One round threshold ecdsa with identifiable abort," *Cryptology ePrint Archive*, 2020.

[6] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergaard, "Fast threshold ecdsa with honest majority," *Journal of Computer Security*, vol. 30, no. 1, pp. 167–196, 2022.

[7] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1051–1066.

[8] C. Komlo and I. Goldberg, "Frost: Flexible round-optimized schnorr threshold signatures," in *Selected Areas in Cryptography*, O. Dunkelman, M. J. Jacobson, Jr., and C. O'Flynn, Eds. Cham: Springer International Publishing, 2021, pp. 34–65.

[9] S. Ricci, P. Dzurenda, R. Casanova-Marqués, and P. Cika, "Threshold signature for privacy-preserving blockchain," in *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum*, ser. Lecture Notes in Business Information Processing, A. Marrella, R. Matulevičius, R. Gabryelczyk, B. Axmann, V. Bosilj Vukšić, W. Gaaloul, M. Indihar Štemberger, A. Kő, and Q. Lu, Eds. Springer International Publishing, pp. 100–115.

[10] A. Abidin, A. Aly, and M. A. Mustafa, "Collaborative Authentication Using Threshold Cryptography," in *Emerging Technologies for Authorization and Authentication"*, A. Saracino and P. Mori, Eds. Cham: Springer International Publishing, 2020, pp. 122–137.

[11] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder, "Roast: Robust asynchronous schnorr threshold signatures," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2551–2564. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3548606.3560583

[12] A. González, H. Ratoanina, R. Salen, S. Sharifian, and V. Soukharev, "Identifiable cheating entity flexible round-optimized schnorr threshold (ice frost) signature protocol," Cryptology ePrint Archive, Paper 2021/1658, 2021, https://eprint.iacr.org/2021/1658. [Online]. Available: https://eprint.iacr.org/2021/1658

[13] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 30:1–30:15. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190538

[14] K. Wüst and A. Gervais, "Do you Need a Blockchain?" in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 45–54.

[15] S. M. H. Bamakan, A. Motavali, and A. Babaei Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, vol. 154, p. 113385, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420302098

[16] N. Szabo, "Formalizing and Securing Relationships on Public Networks," *First Monday*, vol. 2, no. 9, Sep. 1997. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/548

[17] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, 2014.

[18] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1179–1194.

[19] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "Uc non-interactive, proactive, threshold ecdsa with identifiable aborts," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1769–1787.

[20] M. Nystrom and B. Kaliski, "Pkcs# 10: Certification request syntax specification version 1.7," Tech. Rep., 2000.

[21] J. Dreyer, M. Fischer, and R. Tönjes, "Performance analysis of hyperledger fabric 2.0 blockchain platform," in *Proceedings of the workshop on cloud continuum services for smart IoT systems*, 2020, pp. 32–38.

[22] S. Shalaby, A. A. Abdellatif, A. Al-Ali, A. Mohamed, A. Erbad, and M. Guizani, "Performance evaluation of hyperledger fabric," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE, 2020, pp. 608–613.