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

Khattat, Mostafa ; Kromes, Roland

DOI
10.1109/ICBC59979.2024.10634347

Publication date
2024

Document Version
Final published version

Published in
2024 IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2024

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

1st Mostafa Khattat
Delft University of Technology
Delft, Netherlands
m.khattat@etu.tudelft.nl

2nd Roland Kromes
Delft University of Technology
Delft, Netherlands
r.g.kromes@tudelft.nl

2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 979-8-3503-1674-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICBC59979.2024.10634347

***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

Unlike multi-signature schemes, the threshold signatures, also known as "t-of-n" signatures 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 the 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.

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. The integration of threshold signatures into blockchain technology is also advantageous for reinforcing digital identification. Hence, a blockchain-based ecosystem could 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. **The contributions of our work are as follows**: *(i)* Comparison of recently proposed threshold signature schemes. *(ii)* Deployment of APIs on the top of Hyperledger Fabric SDK to enable the transaction signing with FROST signatures algorithm, the registration for a group of devices to the Hyperledger Fabric network and an IoT-enabled Android-based application combining FROST with Hyperledger Fabric SDK. *(iii)* Testing FROST signature verification on Hyperledger Fabric v3.0

The structure of our article is as following: Sect. II provides a comparison on threshold signatures found in the related work. Our proposed protocol is presented in Sect. III. The experiments and results of the proposed solution are showcased in Sect. IV. Finally, Sect V concludes our study.

II. Related Work

In the followings, we classify the threshold signatures found in related work by 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 setting 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 means 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 primarily for the purpose of application in blockchain and cryptocurrency contexts, and one of them has been used in IoT-blockchain-based systems. Table I summarizes the discussed threshold signatures found in the related work, highlighting their features and providing their fair comparison.

III. 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 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 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: *(i)* There exist n users, denoted as

TABLE I: Summary of threshold signatures and their characteristics

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

$u_1, u_2, \ldots, u_n$, 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. *(ii)* 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. *(iii)* A secure communication channel exists among the users that allows for broadcasting or sending a message between any two users. *(iv)* 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. *(v)* 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 the default procedure, Fabric Certificate Authority (CA) generates the private-public key pair for a new user. However, in our approach, it 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 [9]. 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 to the admin CA. In the 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 [10]: *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 $\text{FROST}_{\text{CSR}}$ 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 1, the registration and enrollment process begins with a user delegated by a group of users, and this user is denoted as u_d. The delegated user runs our $\text{FROST}_{\text{CSR}}$ 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 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.

C. Transaction generation and signing API

In the context of our proposed protocol depicted in Figure 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.

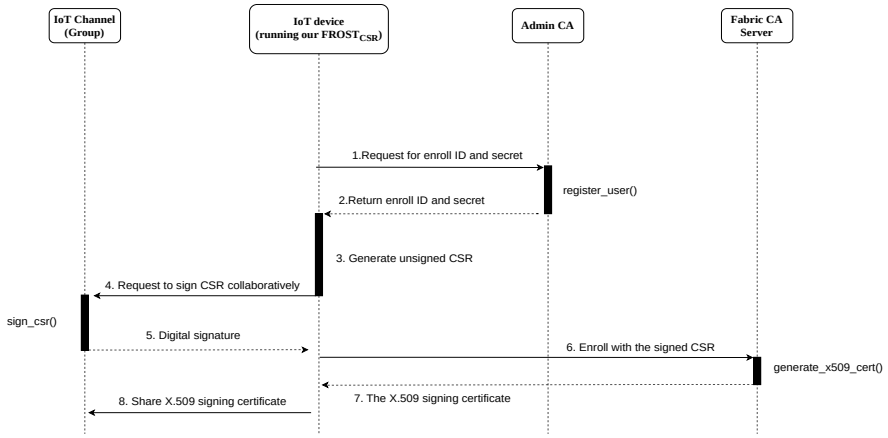

Fig. 1: The registration and enrollment flow

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 2 shows the transaction flow.

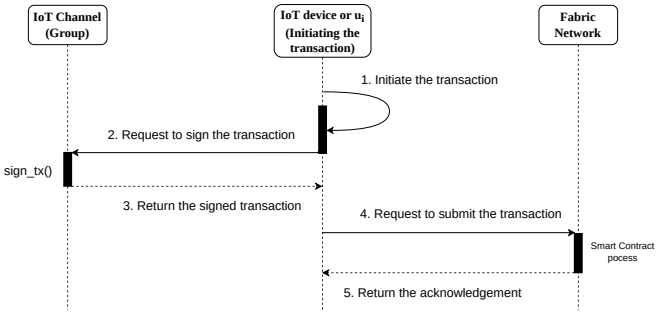

Fig. 2: The transaction flow

D. Technical details about the APIs

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 the user 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 transmit pass messages from the 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 3 shows an overview of all components used in the implementation. Note that all IoT devices follow the same structure. Further, in Figure 3 we used Android as an example of an IoT device.

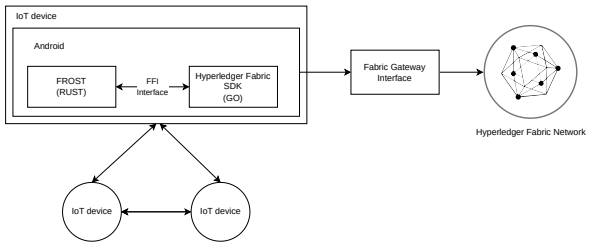

Fig. 3: The Architecture Overview

IV. Experiments and Results

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

A. Setup and Methodology

For the experiments 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 have been compiled and executed natively on the respective devices. To measure the current consumption we used the Otii Pro power meter with a sample rate of $50ksps$.

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. We measured the execution time and energy consumption of FROST, DKLS, and GG20 signatures to highlight their usability in an IoT setting.

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 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. In the experiments, 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.

B. Results

Table II shows a comparison of the energy consumption of the three (FROST, DKLS, GG20) threshold signature schemes with four participants. All of the compared schemes were implemented in Rust programming language. Notably, FROST demonstrates significantly lower energy consumption compared to DKLS and GG20, implying more suitability for IoT use cases.

TABLE II: Energy Consumption of various threshold signatures with 4 participants

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

Figure 4 illustrates the results for the first experiment, measuring the performance of both key generation and signing algorithms across varying threshold configurations. The key generation algorithm shows a relatively consistent performance across various threshold configurations and the number of devices, with an 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.

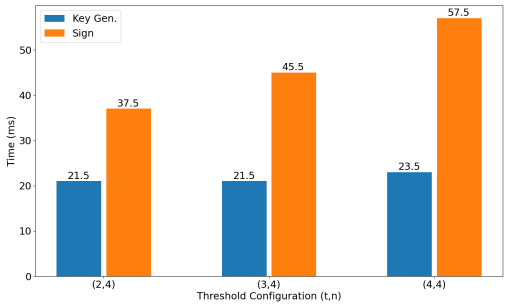

Fig. 4: Running times of FROST

Figure 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.

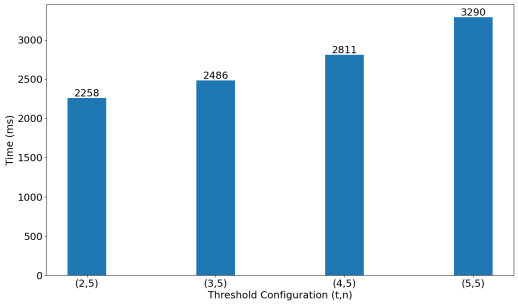

Fig. 5: Transaction submission time using FROST within the Hyperledger Fabric

V. Conclusion

In conclusion, we can affirm that the registration and enrollment are enabled with our proposed $\text{FROST}_{\text{CSR}}$ API, alongside 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. We can conclude that IoT and Hyperledger Fabric blockchain technologies are now "**Completely FROST-ed**" since FROST can be computed with reasonable latency and energy consumption. Our implementation is open-source and can be found at GitHub *https://github.com/mkhattat/Frost-Fabric-IoT*.

ACKNOWLEDGMENT

This work was partly supported by European Union's Horizon Europe research and innovation programme under grant agreement No. 101094901 (SEPTON) and European Union's Horizon 2020 research and innovation programme under grant agreement No. 101021727 (IRIS).

REFERENCES

[1] 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, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie *et al.*, Eds. ACM, 2018, pp. 1837–1854. [Online]. Available: https://doi.org/10.1145/3243734.3243788

[2] R. Gennaro and S. Goldfeder, "One Round Threshold ECDSA with Identifiable Abort," *IACR Cryptol. ePrint Arch.*, p. 540, 2020. [Online]. Available: https://eprint.iacr.org/2020/540

[3] I. Damgård *et al.*, "Fast threshold ECDSA with honest majority," *J. Comput. Secur.*, vol. 30, no. 1, pp. 167–196, 2022. [Online]. Available: https://doi.org/10.3233/JCS-200112

[4] J. 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. [Online]. Available: https://doi.org/10.1109/SP.2019.00024

[5] R. Canetti *et al.*, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts," *IACR Cryptol. ePrint Arch.*, p. 60, 2021. [Online]. Available: https://eprint.iacr.org/2021/060

[6] S. Ricci *et al.*, "Threshold Signature for Privacy-Preserving Blockchain," in *Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum - BPM 2022 Blockchain, RPA, and CEE Forum, Münster, Germany, September 11-16, 2022, Proceedings*, ser. Lecture Notes in Business Information Processing, A. Marrella *et al.*, Eds., vol. 459. Springer, 2022, pp. 100–115. [Online]. Available: https://doi.org/10.1007/978-3-031-16168-1_7

[7] T. 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*, H. Yin *et al.*, Eds. ACM, 2022, pp. 2551–2564. [Online]. Available: https://doi.org/10.1145/3548606.3560583

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

[9] C. Komlo and I. 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*, ser. Lecture Notes in Computer Science, O. Dunkelman *et al.*, Eds., vol. 12804. Springer, 2020, pp. 34–65. [Online]. Available: https://doi.org/10.1007/978-3-030-81652-0_2

[10] M. Nyström and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7," *RFC*, vol. 2986, pp. 1–14, 2000. [Online]. Available: https://doi.org/10.17487/RFC2986