

A SURVEY OF FROST AND ITS VARIANTS, RFC9591 INSTANTIATIONS, IMPLEMENTATION AND EFFICIENCY ANALYSIS

***Ji-Jian Chin^{1*}, Syh-Yuan Tan², Sook-Chin Yip³, Nur Hanis Sabrina, Suhaimi⁴, Raadhesh Kannan¹
and Hai-Van Dang¹***

¹School of Engineering, Computing and Mathematics,
Universiti of Plymouth,
Plymouth, United Kingdom.

²Faculty of Information Science and Technology,
Multimedia University,
Melaka, Malaysia.

³Faculty of Artificial Intelligence and Engineering,
Multimedia University,
Cyberjaya, Malaysia.

⁴Faculty of Information Science and Technology,
Universiti Kebangsaan Malaysia,
Bangi, Malaysia.

Emails: ji-jian.chin@plymouth.ac.uk^{1*}, sytan@gmail.com², scyip@mmu.edu.my³,
sabrinasuhaimi@ukm.edu.my⁴, raadhesh.kannan@plymouth.ac.uk¹, hai-van.dang@plymouth.ac.uk¹

ABSTRACT

Threshold signature schemes enable a group of participants to collaboratively generate a signature if a subset of t out of n participants willingly participate in the signing process. Among recent developments, FROST (Flexible Round-Optimised Schnorr Threshold signatures) stands out for its efficiency, flexibility, and strong security guarantees in both theoretical and applied settings. This survey provides a comprehensive overview of FROST, starting from its cryptographic foundations in Schnorr signatures and the discrete logarithm problem to its formal security models and threat assumptions, including the unique robustness properties of FROST in the presence of malicious or aborting signers. We examine its two-round signing protocol, distributed key generation (DKG), and integration with secure multiparty computation frameworks. Furthermore, the survey categorises and analyses key extensions and adaptations of FROST, such as proactive and dynamic group configurations, post-quantum variants, compatibility with verifiable secret sharing, and deployments over alternative curves and signature primitives. We also evaluate practical implementations, standardisation efforts (e.g., by the IRTF CFRG), and real-world applications in blockchain and distributed systems. Finally, we also provide a comparative analysis of FROST instantiations according to RFC9591 specifications, showing concrete efficiency analysis results. By mapping out current research and highlighting open challenges, this survey aims to guide future work in robust, scalable threshold signature protocols anchored in the FROST paradigm.

Keywords: Threshold Signature; Survey; Implementation; Standards; Security Models; RFC9591; Simulator; Runtime

1. INTRODUCTION

Threshold cryptography is introduced so that multiple parties owning sections of a key share can participate in a cryptographic activity, such as encryption or signing. The threshold signature was first introduced by Desmedt and Frankel in [1] using the RSA secret sharing method first introduced by Shamir in [2]. Since its inception, threshold signatures have seen various development over the decade such as extensions into the identity-based setting [3], aggregate signatures [4], and more recently in blockchain [5] and IoT applications [6], where it is desirable to have multiple parties actively collaborating to perform signing.

To achieve the signing, there are two methods of construction: one where t -out-of- n parties need to convene to reconstruct their keys before signing as the original proposal by Shamir [2], or one that allows t -out-of- n signing without reconstructing keys. FROST, a threshold signature scheme by Komlo and Goldberg first introduced in [7], falls in the latter category: it is constructed as a threshold Schnorr [8] signature protocol designed to minimise signing rounds. Each signing session takes two rounds in its standard form:

- 1) a commitment broadcast and
- 2) signature shares. FROST adds a pre-processing phase which allows for an effective single-round signature, thus improving efficiency during signing.

Since its introduction, FROST has received significant attention. Improvements were suggested in FROST2 [9] and most recently a full adaptive proof of security in [10]. Other variants such as ROAST[11], ICE-FROST[12], rerandomised FROST [13], and OLAF [14] offer additional security features. Still, each comes with tradeoffs that will be discussed in more depth later in the paper. The main goal of this work is to provide a one-stop reference point for all developments for FROST since its inception five years ago, written in a form accessible to more general cybersecurity audiences not necessarily trained in cryptographic writing. It should be mentioned that this work was prepared in conjunction with the commissioning of MySEAL 2.1 [15], an initiative by the Malaysian National Cyber Security Agency to review cryptographic primitives suitable for government use. Thus, these summative findings for FROST as one of the threshold signatures for consideration in the MySEAL 2.1 framework are written in a manner for decision-makers to understand.

The rest of the paper is structured as follows. Section 2 provides the definitions for the security models that FROST and its extensions use and then proceed with preliminaries like model and assumptions in Section 3. We describe threshold signatures in detail in Section 4 and provide the details of FROST scheme description in Section 5. We then summarise the extensions that have been built based on FROST in recent years in Section 6. We discuss the standardisation document RFC9591 for FROST in Section 7, describing in more detail instantiations of FROST that use different curves featured in that document. Section 8 will showcase our experiment running the FROST instantiations in RFC9591 in Python and provide some concrete runtime results. We conclude with a summary and open problems in Section 9.

2. METHODOLOGY

We use PRISMA in our survey. For identification, in our search terminology, we use the keywords “Threshold Signature” and “FROST”. We performed our searches primarily for academic paper results for the theoretical findings. We exclude everything not FROST related, as our main scope of study was only for MySEAL 2.1’s FROST signature at this point of writing[15]. Secondary searches using the same terminology with the additional keywords of “implementation”, “standard”, “blockchain” and “IoT” were added to Google Scholar searches for implementations and standardisation. The secondary search revealed the recent IETF RFC document [16], which warranted FROST for further investigation under MySEAL 2.1 [15].

For our academic paper review, we started our search at eprint.iacr.org, which yielded 18 results. While most of the results are recent variants that we consider, several made minor mentions of FROST in part of their work. We followed up with a search in Google Scholar combining the keywords “Threshold Signature FROST”, which yielded more accurate matches, as “FROST” alone produced results in geological snow studies. At the same time, “threshold signature” alone showed other threshold signatures not related to FROST. We picked up on the implementation papers from this Google search. However, we exclude these other threshold signature schemes as only FROST has an IETF RFC document, which makes it eligible for MySEAL consideration.

Finally, we used Artificial Intelligence (AI) in Perplexity AI and ChatGPT Deep Research modes for additional search and analysis. We used the following prompt for Perplexity AI and ChatGPT: “Produce a review on the threshold signature FROST and all of its extensions. Review them based on the security properties, adversarial models, and implementations on different curves. Proceed to discuss the different use cases for FROST and its extensions in blockchains and other case studies. Finally, discuss the standardisation process that was recently approved.”

We report that both AI platforms produced shallow levels of research, showing the main points of the matches we found on eprint.iacr.org, but did not delve in depth into security models or assumptions. However, it did manage to highlight the benefits and trade-offs that we built TABLE 2 on. There were also a few cases of hallucination and misinformation, citing papers that did not exist or papers that had a wrong reference link, most notoriously the genesis paper by Desmedt and Frankel [1] was attributed to a textbook by Desmedt produced in 2001 instead of the original paper, and another incident where ROAST was confused as a different scheme to FROST3. Still, it turns out FROST3 was a re-labeling by another paper, referencing it as such.

Therefore, while AI can augment analysis work, a human-in-the-loop verification process must be used to check that the results are accurate. The authors do report that AI helped produce a summary analysis of the improvements

and tradeoffs between the FROST variant. However, we took precautions to confirm the facts from the references ourselves. Our final survey includes the original FROST paper, as well as extensions from Table 2.

Finally, we note that there are no existing surveys on FROST – the closest review paper we found on the matter was a non-peer reviewed whitepaper by SQRR Research [17].

3. SECURITY MODEL AND ASSUMPTIONS

Intractable Assumptions

Definition 1: The Discrete Logarithm (DL) problem is believed to be hard to solve. When given a number g and g^x computed using a secret x , it is assumed to be difficult to compute x .

Definition 2: The One-More Discrete Logarithm (OMDL) problem is an interactive analog of the DL problem. As its basis, it is similar to DL, that given g and g^x computed using a secret number x , it is assumed to be difficult to compute x . With the one-more assumption, the attacker is given access to two oracles: a challenge oracle and a solution oracle. The goal is to get $n + 1$ number of challenges $g_0^{x_0}, \dots, g_n^{x_n}$ but query for only n solutions for $x_1, \dots, x_n$, that is, the attacker needs to solve the challenge $g_0^{x_0}$ by producing x_0 by itself.

The OMDL problem is generally used for interactive proofs, where solutions are required to be given to an adversary in the proof while the simulator does not know the answers. However, it is treated as a stronger assumption than the static classical DL problem.

Furthermore, recent proofs for FROST have assumed this to be in the algebraic group model [18]. In the algebraic group model, each element must be expressed as a combination of the initial used group elements. Still, at times, this assumption that the adversary knows what the initial group elements or the relationships between them are to form combinations is often a very strong assumption and is therefore not desirable. It is only used when there is no evident solution for a proof found under the standard group model, where the adversary can use any strategy on group encodings, and the generic group model [19], where the adversary selects randomly encoded objects.

4. THRESHOLD SIGNATURE SCHEMES

A threshold signature scheme allows a group of n participants to jointly sign a message so that at least t of them must cooperate to produce a valid signature. This is useful for security and fault tolerance in systems where multiple parties need to approve actions. A t -out-of- n threshold signature scheme consists of the following algorithms:

1. SETUP generates a common public parameter based on a security level parameter input.
2. KEYGEN is run by each participant, where they create a private key and contribute to a shared public key. This public key can be used to verify future signatures made by a qualified subset of participants.
3. PREPROCESS is where each participant prepares for a future signing event by generating temporary values called “pre-signatures” that can be reused.
4. SIGN takes the private key, shared public key, pre-signature and state information, and a message to be signed, and generates a partial signature.
5. COMBINE takes in the shared public key, pre-sign state information, signature shares from SIGN and the message, and combines them into a single signature. If a partial signature is invalid, it is rejected, and the erroneous signer can be identified.
6. VERIFY takes in the shared public key, message and signature and allows anyone to verify the signature is valid.

Security Properties

- *Correctness*: If at least t honest participants follow the signing protocol, a valid signature will be produced with high probability.
- *EUFCMA Security (Unforgeability)*: An attacker controlling fewer than t corrupted participants cannot forge a valid signature on a new message that was not signed before, even if they are able to trick honest signers to partially sign it.

5. THE FROST SIGNATURE SCHEME

The FROST scheme (Flexible Round-Optimised Schnorr Threshold signatures) [7] is built using the threshold signature model above. It does not require a shared secret amongst participants, making it a *t-out-of-n* threshold signature scheme rather than a shared secret threshold signature. FROST runs signing in two stages: a set up stage (which includes SETUP, KEYGEN and PREPROCESS) and a signing stage (which includes SIGN, COMBINE, and VERIFY). These are described below in more detail:

Table 1: Frost threshold signature scheme description

FROST Setup and Key Generation Protocol In the first round, each participant creates a random secret polynomial that helps to securely share secret information with others. Each participant then generates public values called “commitments” to show that they are being honest about their polynomial. They prove they know the secret starting value of their polynomial using a cryptographic proof (i.e., a variant of the Schnorr proof). Each participant verifies these proofs. If any proof fails, the setup stops. In the second round, each participant sends parts of their polynomial secret shares to all other participants. Recipients verify that the shares match the public commitments. Each participant then computes their long-term secret share by combining all their received shares. The corresponding shared public key for the whole group is derived by multiplying the starting terms from the polynomials of every participant.
FROST Pre-processing Protocol Each P_i decides how many nonces (temporary secret values) to generate for future signing. Each nonce is paired with a random number, and their corresponding public commitments are published. These nonces are stored for use in the actual signing phase.
FROST Signing and Combining Protocol A coordinator (also known as the Signature Aggregator, SA) chooses at least t participants to sign a message. The SA collects one nonce pair from each selected participant and bundles these values into a “binding list” to be sent to all signers along with the message to be signed. Each signer verifies the commitments and the message. If valid, they each compute a special coefficient that binds the message to their commitment. Using this special coefficient, they then compute their individual and group commitments. Subsequently, a challenge value is calculated based on the group commitment, the public key, and the message. Participants compute a Lagrange coefficient ensuring each share has the correct weight. Then, each participant will produce a response using their nonces, secret share, and challenge. It is worth noting that used nonces are deleted to avoid reuse, as it will be a security risk. Each response is verified to ensure correctness. If all are valid, the SA sums the responses into z and outputs the final signature (R, z). Any invalid responses will cause an abort, and the protocol needs to restart.
FROST Verification The aggregated signature is verified using a check similar to the Schnorr signatures.

Theorem 1: (EUF-CMA[7])

If solving the discrete logarithm problem is hard in group G , then the FROST signature scheme is secure against forgery. Specifically, if an attacker tries to forge a signature while controlling fewer than t participants, the chance of success is negligible. The security parameters would be: how many queries the attacker can make, how many messages are preprocessed, how many signings happen, and the computational cost of operations like exponentiation in the group.

6. EXTENSIONS

In this section, we show the various variants of FROST that have been proposed over the last five years since its proposal. However, to first understand the motivations behind the various extensions to FROST, we look at some of the drawbacks of the original proposal. These include:

1. Dependence on pre-processing: The pre-processing phase of FROST causes latency in online settings and makes pre-signing or batch signing harder without reliable coordination.
2. Lack of proactive security: Since FROST is a t -out-of- n threshold signature, there is no proactive secret sharing and therefore it cannot periodically refresh shares, recover from compromised shares, nor rotate participants securely. This may affect long-term deployments, where key compromise may happen, but there is no way to recover.
3. Static participants: FROST assumes a fixed set of participants after setup. There is no support for dynamic joins or removals of signers once the system is set up. This may not be sufficient for dynamic applications like e-wallets, blockchains or decentralised identity management systems.
4. No accountability: There is no public blame mechanism in the original FROST, and the signing merely aborts when there are misbehaving participants.
5. Assumes ideal distributed key generation or trusted setup: There is no specified distributed key generation (DKG) protocol in the original proposal, and therefore FROST assumes either the trusted dealer distributes shares securely or an external DKG is used.
6. Static corruption model in security proof: Adaptive adversaries are not modelled in the original FROST, where the adversary can choose participants to corrupt in the security game. The security game of the original FROST assumes a fixed set of corrupted participants.
7. Not post-quantum secure: As it relies on discrete logarithms for its security guarantees, this breaks down in the presence of a quantum computer.

Therefore, the variants in Table 2 were introduced to circumvent these drawbacks.

Table 2: Variants of FROST and their characteristics

Variant	Year	Benefits	Tradeoffs
FROST2[9]	2021	FROST2 is an optimised variant that aims to improve signing efficiency of the original FROST by reducing the number of exponentiations required to compute a group element 'R' for the number of signers to a constant, where only one exponentiation is required for each signer. There is a duplicate nonce check to aid in the proof, but overall, FROST2 was meant to be an efficiency improvement.	According to Bellare et al.'s work [20] on reexamination of threshold signature proofs of security, FROST achieves TS-SUF-3, but FROST2 achieves TS-SUF-2 only; therefore, there is a tradeoff in security guarantees when compared to the original FROST.
ROAST[11]	2022	Robust Asynchronous Schnorr Threshold Signatures (ROAST), which is also identified as FROST3 by Chu et al., includes another	As it is asynchronous, multiple concurrent FROST signing sessions can be conducted, but this leads to a higher overall computational burden for the signers and

Variant	Year	Benefits	Tradeoffs
		optimisation on the communication front, where in the pre-processing phase allows for aggregation of pre-signature shares and broadcasting them instead of conducting them in a linear fashion.	coordinator than the complexity of a single successful FROST signing operation. ROAST inherits its security from FROST and it is therefore also TS-SUF-3.
OLAF[14]	2023	OLAF is a combination of FROST3/ROAST with a distributed key generator (DKG) called SimplPedPoP, a simplified Pederson DKG variant to provide unforgeability that avoids reliance on the Algebraic Group Model. This provides more reliable proof for practitioners that avoids complex, idealised models.	The focus of OLAF is unforgeability in the DKG setting under ROM. Robustness of wrapper protocols such as the one in ROAST may be applied.
ICE-FROST[12]	2023	Identifiable Cheating Entity (ICE)-FROST improves the robustness over FROST (which aborts if there are misbehaviors) by allowing the protocol to continue despite misbehaving participants. ICEFROST introduces mechanisms to allow the protocol to continue as long as there are sufficient honest participants and to identify misbehaving parties.	The cost of ICE-FROST comes in the form of additional operations overhead, which includes encryption, broadcasting of shares and the complaint management protocol in the key generation phase.
Rerandomised FROST[13]	2024	Rerandomised-FROST introduced the idea of rerandomisable public and private keys for enabling unlinkability in privacy-focused applications, particularly for ZCash. Unforgeability is then proven under the algebraic one-more discrete logarithm assumption (A-OMDL) in the random oracle model.	There is significant overhead in signing and aggregation performance due to an additional elliptic curve point multiplication for rerandomisation. Also not explicitly mentioned are the robustness properties, likely to follow FROST's original abort nature in the presence of misbehaving participants.
D-FROST[20]	2024	D-FROST or Dynamic-FROST merges FROST with CHURP. CHURP is a dynamic and proactive secret sharing scheme. This combination provides for dynamic committee formations and changes to the number of participants in the threshold signature without having to alter the group's public key and is used primarily in cryptocurrency wallets that need to update signers without moving funds.	There is additional operational overhead from CHURP as it is an additional protocol over the original FROST. CHURP also relies on the $(t-1)$ -Strong Diffie-Hellman and q -PKE assumptions, which are not well-studied assumptions, so there is a decrease in security confidence.

In addition to augmentations to the original FROST, some address the proof of security status for FROST. In 2022, Bellare et al. in [21] proposes unforgeability criteria with linearly increasing strengths, beginning with TS-UF-0 up to TS-UF-4, as well as strong unforgeability notions TS-SUF-0 up to TS-SUF-4. The authors showed that the original FROST is TS-SUF-3-secure, while FROST2 only achieves TS-SUF-2 assuming the OMDL problem is hard. Later, Cremers et al. [22] reveal a model linearity issue in the unforgeability criteria by showing that TS-(S)UF-3 is not weaker than, but independent from TS-(S)UF-4. They suggest to upgrade TS-(S)UF-4 as a solution besides showing that the upgraded version is equivalent to their proposed security model SiGu-4:aLRhPP. Instead of adopting Cremer et al.'s approach, Bellare et al. [21] took another alternative to downgrade TS-SUF-3 and resume the security linearity.

While the security notions are improved, it is still an open problem to achieve a TS-SUF-4-secure variant of FROST with better efficiency, other than their transformation proposed using their Theorem 2 that uses an echo scheme. Tan et al. most recently proposed a technique to boost FROST2 to achieve this[23].

On the contrary, Crites et al. brought to attention the issue of adaptive adversaries in [24], showing all variants of FROST cannot be proven secure against adaptive adversaries without modifications or assuming the hardness of a search problem. The subsequent work [25] answers the open problem by Crites et al., where they showed adaptive security for FROST using the algebraic OMDL in the random oracle model, in conjunction with the low-dimensional vector representation (LDVR) problem. However, LDVR is used for the sole purpose of this proof and is not presented anywhere else. The authors do show that adaptive security of FROST is achievable for $t/2$ malicious parties under these assumptions without further modifications.

One final entry which we acknowledge but does not fall under the focus of our survey is FlexHi FROST from [26], as this would expand the works towards hierarchical threshold signatures, and introduces a lot more complexity beyond the scope of our study of standard threshold signatures. We conclude this section with a mention that research is still active and ongoing to address the benefits and trade-offs for FROST.

7. RFC9591 AND INSTANTIATIONS

In 2024, FROST was published in the Internet Engineering Task Force (IETF)'s RFC9591 under the Crypto Forum Research Group (CFRG) [16] as an informational Request for Comment (RFC). In this document, there are explicit details on proper implementation for FROST specified so FROST can be more broadly adopted, rather than remain in the theoretical cryptography domain. In the RFC, there are fully specified ciphersuites that cover Ed25519, Ed448, P-256, Ristretto255, secp256k1 with their corresponding hash functions, as well as test vectors for each ciphersuite to validate the implementation correctness. Moreover, specifications for operational guidance are provided, addressing issues such as nonce reuse, side-channel resistance and coordinator-less and coordinator-based modes with minimal trusting assumptions. Due to this RFC, FROST is one of the primary considerations for MySEAL 2.1.

In this section, we look at the implementation specifications outlined in RFC9591 in more detail. FROST is constructed using two main cryptographic components: an EC-based prime-order group and a cryptographic hash function. We examine the Ed25519, Ed448, P-256, Ristretto255 and secp256k1 implementations.

FROST (Ed25519,SHA-512)

As specified in RFC8032[25], the elliptic curve Ed25519 is a twisted Edwards curve over a 255-bit prime field, specifically $2^{255} - 19$, that uses the equation $ax^2 + y^2 = 1 + dx^2y^2$ with proper chosen coefficients, thus offering high performance and strong security properties. As part of the Edwards-curve Digital Signature Algorithm EdDSA family, specifically designed for deterministic digital signatures, Ed25519 provides an equivalent of 128-bit security. As outlined in RFC8032, Ed25519 demonstrates resistance against side channels such as timing attacks due to its fixed-time implementation characteristics. Ed25519 is deployed in various applications such as TLS, SSH, and multiple cryptocurrency systems [27] due to its high performance. RFC9591 lists Ed25519 as one of the ciphersuites for FROST, to be used with the SHA-512 hash function under the contextString parameter of "FROST-ED25519-SHA512-v1".

FROST (Ed448, SHAKE-256)

Also specified by RFC8032[25], Ed448 is a twisted Edwards curve over a 448-bit prime field using the equation $ax^2 + y^2 = 1 + dx^2y^2$, similarly with specific coefficients. Like Ed25519, Ed448 also belongs to the EdDSA family. However, it offers stronger security at approximately 224-bit security compared to Ed25519. It also provides resistance against side-channel attacks. Additionally, RFC9496 [25] describes implementing the decaf448 prime-order group using Ed448 points as internal representations. For FROST, Ed448 ciphersuite is used with SHAKE256 hash function under the contextString "FROST-ED448-SHAKE256-v1".

FROST (P-256, SHA-256)

The elliptic curve P-256 is a prime-order elliptic curve defined over a 256-bit prime field and constructed under the equation $y^2 = x^3 + ax + b$. The National Institute of Standards and Technology (NIST) standardised it under the FIPS 186-4 document[28], with a brief mention in the most recent update in FIPS 186-5[29] that supersedes it. P-256 provides 128-bit security and is widely used in protocols like TLS, ECDSA, and ECDH. It is a preferred choice for resource-constrained environments due to its smaller key size and computational efficiency[30]. In

FROST, it is used with the SHA-256 hash function under the contextString parameter of “FROST-P256-SHA256-v1”.

FROST (Ristretto255, SHA-512)

The elliptic curve Ristretto255, specified by RFC 9496 [28], is a prime-order group constructed from the Ed25519 elliptic curve. While Ed25519 is a twisted Edwards curve defined over a 255-bit prime field, Ristretto255 provides a method to map elements from the underlying curve into a prime-order group, thereby eliminating cofactor issues that can lead to subtle vulnerabilities in cryptographic protocols. This construction ensures that all operations result in valid points of the prime-order subgroup, simplifying cryptographic protocol design and improving safety. It leverages the efficiency of Ed25519's curve arithmetic while enhancing security guarantees, making it suitable for applications requiring robust and secure group operations, such as password-authenticated key exchange (PAKE) and zero-knowledge proofs [28]. For RFC9591, Ristretto255 ciphersuite is to be used with the SHA-512 hash function, with the value of the contextString parameter for FROST is “FROST-RISTRETTO255-SHA512-v1”.

FROST (Secp256k1, SHA-256)

The elliptic curve Secp256k1 is characterised by the equation $y^2 = x^3 + 7$ with a 256-bit prime order and coefficients $a = 0, b = 7$. It is standardised by the Standards for Efficient Cryptography Group in the SEC 2 specification [30, p. 2] and provides 128-bit security. Secp256k1 gained prominence due to its deployment in Bitcoin as the Elliptic Curve Digital Signature Algorithm (ECDSA)'s curve of choice with efficiency that scales with the size of the blockchain[31]. In FROST, secp256k1 uses the SHA-256 hash function under the contextString parameter of “FROST-secp256k1-SHA256-v1”.

8. IMPLEMENTATION AND CONCRETE RUNTIMES

We have implemented the FROST threshold signature with different elliptic-curve based signature and hash algorithms as mentioned in section 7 using Python 3.14.2 (though the code is compatible with Python 3.6.2 and above) and the ecdsa and PyNaCl libraries. The github repository can be found at [32]. The experiments were run in the computer with Intel® Core™ i7-6700 CPU @ 3.40 GHz and 16 GB RAM. Regarding the settings, the number of participants is fixed at 10 and the threshold value is 5. Each variation implementation is executed 30 times and the runtime of the following functions and the total runtime for all participants are recorded in milliseconds (ms): Setup and Key Generation, Pre-processing per participant and all participants, Signing per participant and all participants, Combining, Verification (see Table 3).

Based on the recorded data, we summarize the statistics values as below:

Table 3: Runtime of variants of FROST

Phase	FROST (Ed25519, SHA-512)			FROST (Ed448, SHAKE-256)			FROST (P-256, SHA-256)			FROST (Ristretto255, SHA-512)			FROST (Secp256k1, SHA-256)		
	mean	median	std	mean	median	std	mean	median	std	mean	median	std	mean	median	std
Setup	33.72	33.72	1.42	38.76	38.76	7.11	40.53	40.53	1.42	39.19	39.19	1.42	38.31	38.31	0.00
Key generation	142.09	141.76	2.92	3213.18	3194.32	125.55	1546.78	1517.02	101.34	1782.94	1750.36	101.34	4079.21	4052.08	89.94
Preprocess	0.378	0.354	0.070	86.042	85.522	7.437	39.406	36.558	9.032	34.681	33.541	9.032	92.684	92.723	2.391
signing	0.019	0.015	0.013	0.030	0.027	0.011	0.023	0.020	0.008	0.021	0.020	0.008	0.022	0.017	0.015
combining	1.771	1.730	0.105	51.188	49.648	5.064	22.178	21.187	3.344	24.639	23.551	3.344	47.624	47.219	2.379
verifying	0.254	0.243	0.044	18.091	17.315	2.366	7.725	7.593	0.876	8.300	7.691	0.876	19.300	18.907	1.607
preprocess per participant	0.075	0.070	0.014	17.205	16.262	1.487	7.878	7.350	1.806	6.933	6.663	1.806	18.535	18.424	0.478
signing per participant	0.004	0.003	0.002	0.005	0.004	0.002	0.004	0.003	0.002	0.004	0.003	0.002	0.004	0.003	0.003

Among the phase, the signing, combining and verifying run more often than the setup and key generation. Therefore, we present the boxplot diagrams of these phases in Figure 1 so that we can have insights into the comparison between the different variants. The signing runtime per participant (Figure 1b) and all participants (Figure 1a) of the variants FROST (Ed25519,SHA-512) and FROST (Secp256k1, SHA-256) are much lower than

the others. Meanwhile, only FROST (Ed25519,SHA-512) has the lowest runtime in combining and verifying phases. In total, FROST (Ed25519,SHA-512) has the best performance.

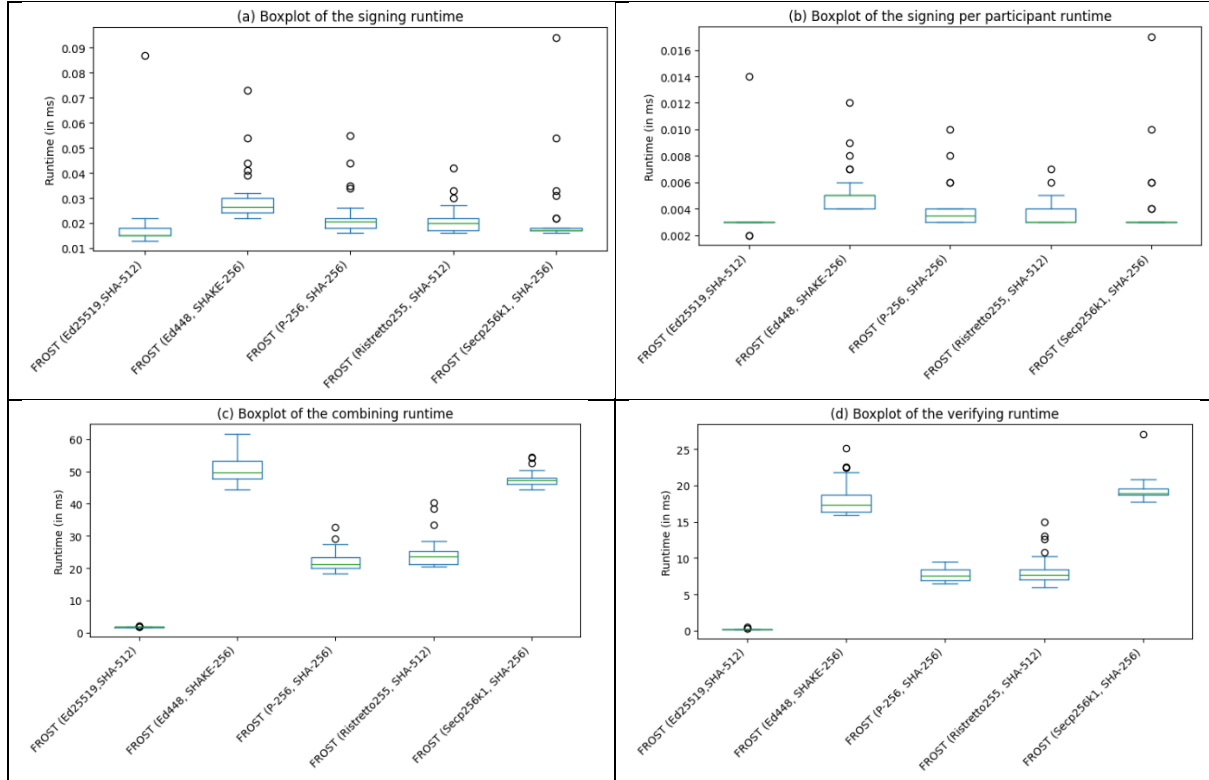


Fig. 1: Boxplots of runtime in different phases

Looking closely at FROST (Ed25519,SHA-512) in Table 3, the standard deviation (std) values of all the phases except “signing per participant” are low compared to the mean values whilst the median is quite close to mean. This demonstrates the stability in the experiments. Regarding “signing per participant”, although the standard deviation is not low compared to mean value, the fluctuation is not much due to the low values of the runtime (min=0.002ms, max=0.014ms). In general, the experiments outputs are stable.

When considering the runtime among the different phases, as in Table 3, for all the variants except FROST (Ed25519,SHA-512), Key Generation is the most time consuming, followed by Setup phase, Preprocess and Combining. The two phases Setup and Key Generation are not run regularly but Preprocess and Combining can run regularly. Fortunately, the variant FROST (Ed25519,SHA-512) only have Setup and Key Generation time consuming whilst the other phases runtime are very low.

Based on the experiments analysis which shares the same pattern with the Rust implementation by Zcash Foundation [33], we can recommend the use of FROST (Ed25519,SHA-512) for better performance compared to the others variants.

9. CONCLUSION AND FUTURE WORK

In this survey, we examined the FROST threshold signature, discussed its drawbacks and how they are addressed by later variants and extensions, each with their own features and trade-offs. Our analysis produced a table that facilitates cost-benefit decisions. Finally, we reviewed the IETF standard approved for FROST, depicting some details on the advised curves and parameters for implementation. We hope this review offers decision-makers who wish to adopt FROST as part of their cybersecurity solution insights to make their adoption decisions, while providing open research problems for interested academics to pursue and improve threshold signatures.

For future work, we will pursue a more stringent approach in implementing FROST in RUST, as well as following client-server architectures. It is also worth noting if tight Schnorr [34] and tight BLS [35] variants of signatures

can be utilized to generate tighter security proofs for FROST-like threshold signatures, and we will be investigating this open problem.

ACKNOWLEDGEMENT

The first and second authors wish to acknowledge the commissioning of the MySEAL 2.1 project which directly led to the results of this work. The third author wishes to acknowledge the Fundamental Research Grant Scheme FRGS/1/2023/ICT07/MMU/03/1 for partly supporting this research. Finally, the fourth author would like to thank Universiti Kebangsaan Malaysia for sponsoring this publication.

REFERENCES

- [1] Y. Desmedt and Y. Frankel, 'Threshold Cryptosystems', in *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, in CRYPTO '89. Berlin, Heidelberg: Springer-Verlag, 1989, pp. 307–315.
- [2] A. Shamir, 'How to share a secret', *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.
- [3] J. Baek and Y. Zheng, 'Identity-Based Threshold Decryption'. 2003. [Online]. Available: <https://eprint.iacr.org/2003/164>
- [4] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, 'Aggregate and Verifiably Encrypted Signatures from Bilinear Maps', in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 416–432.
- [5] ZcashFoundation, 'ZF FROST (Flexible Round-Optimised Schnorr Threshold signatures)'. 2024. Accessed: Nov. 23, 2024. [Online]. Available: <https://github.com/ZcashFoundation/frost/>
- [6] E. Kustov, 'A Lattice-Based Threshold Signature Scheme for IoT Applications', in *2025 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, 2025, pp. 1–5. doi: 10.1109/WECONF65186.2025.11017219.
- [7] C. Komlo and I. Goldberg, 'FROST: Flexible Round-Optimized Schnorr Threshold Signatures', 2020, 2020/852. Accessed: Jun. 16, 2025. [Online]. Available: <https://eprint.iacr.org/2020/852>
- [8] C. Schnorr, 'Efficient signature generation by smart cards', *Journal of Cryptology*, vol. 4, pp. 161–174, Jan. 1991, doi: 10.1007/BF00196725.
- [9] E. Crites, C. Komlo, and M. Maller, 'How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures', 2021, 2021/1375. Accessed: Jun. 16, 2025. [Online]. Available: <https://eprint.iacr.org/2021/1375>
- [10] E. Crites, J. Katz, C. Komlo, S. Tessaro, and C. Zhu, 'On the Adaptive Security of FROST'. 2025. [Online]. Available: <https://eprint.iacr.org/2025/1061>
- [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*, in CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 2551–2564. doi: 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'. 2021. [Online]. Available: <https://eprint.iacr.org/2021/1658>
- [13] C. P. L. Gouvea and C. Komlo, 'Re-Randomized FROST'. 2024. [Online]. Available: <https://eprint.iacr.org/2024/436>
- [14] H. Chu, P. Gerhart, T. Ruffing, and D. Schröder, 'Practical Schnorr Threshold Signatures Without the Algebraic Group Model'. 2023. doi: 10.1007/978-3-031-38557-5_24.
- [15] 'MySEAL Introduction'. Accessed: Jun. 18, 2025. [Online]. Available: <https://mykripto.cybersecurity.my/index.php/services/myseal/introduction>
- [16] D. Connolly, C. Komlo, I. Goldberg, and C. A. Wood, 'RFC 9591 The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures'. RFC Editor, 2024.
- [17] SQRR Research, 'FROST Threshold Signatures', SQRR, Sep. 2023. [Online]. Available: <https://sqrr.xyz/reports/docs/6/2/FROST%20Threshold%20Signatures.pdf>
- [18] G. Fuchsbauer, E. Kiltz, and J. Loss, 'The Algebraic Group Model and its Applications'. 2017. [Online]. Available: <https://eprint.iacr.org/2017/620>
- [19] V. Shoup, 'Lower Bounds for Discrete Logarithms and Related Problems', in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, in Lecture Notes in Computer Science, vol. 1233. Springer, 1997, pp. 256–266. doi: 10.1007/3-540-69053-0_18.

- [20] A. Cimatti *et al.*, ‘Dynamic-FROST: Schnorr Threshold Signatures with a Flexible Committee’. 2024. [Online]. Available: <https://eprint.iacr.org/2024/896>
- [21] M. Bellare, S. Tessaro, and C. Zhu, ‘Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST’, 2022, 2022/833. Accessed: Jun. 16, 2025. [Online]. Available: <https://eprint.iacr.org/2022/833>
- [22] C. Cremers, A. Peltonen, and M. Zhao, ‘An Extended Hierarchy of Security Notions for Threshold Signature Schemes and Automated Analysis of Protocols That Use Them’. 2024. [Online]. Available: <https://eprint.iacr.org/2024/1920>
- [23] S.-Y. Tan, W. Wang, and R. Chow, ‘From TS-SUF-2 to TS-SUF-4: Practical Security Enhancements for FROST2 Threshold Signatures’. 2026. [Online]. Available: <https://eprint.iacr.org/2026/075>
- [24] E. Crites and A. Stewart, ‘A Plausible Attack on the Adaptive Security of Threshold Schnorr Signatures’. 2025. [Online]. Available: <https://eprint.iacr.org/2025/1001>
- [25] S. Josefsson and I. Liusvaara, ‘Edwards-Curve Digital Signature Algorithm (EdDSA)’, no. 8032. in Request for Comments. RFC Editor, Jan. 2017. doi: 10.17487/RFC8032.
- [26] M. A. Bingol, S. Kocaman, A. Dogan, and S. K. Toplu, ‘FlexHi: A Flexible Hierarchical Threshold Signature Scheme’, in *Intelligent Computing*, K. Arai, Ed., Cham: Springer Nature Switzerland, 2024, pp. 509–529.
- [27] M. Hamburg and I. A. Lovecruft, ‘The ristretto255 and decaf448 Groups’, Internet Engineering Task Force (IETF), Request for Comments 9496, Jan. 2024. Accessed: May 24, 2025. [Online]. Available: <https://datatracker.ietf.org/doc/rfc9496/>
- [28] NIST-FIPS186-4, ‘Digital Signature Standard (DSS)’, U.S. Department of Commerce, Federal Information Processing Standard (FIPS) 186-4 (Withdrawn), Jul. 2013. doi: 10.6028/NIST.FIPS.186-4.
- [29] NIST-FIPS186-5, ‘Digital Signature Standard (DSS)’, U.S. Department of Commerce, Federal Information Processing Standard (FIPS) 186-5, Feb. 2023. doi: 10.6028/NIST.FIPS.186-5.
- [30] C. Research, ‘SEC 2: Recommended Elliptic Curve Domain Parameters’. 2000. [Online]. Available: <https://www.secg.org/sec2-v2.pdf>
- [31] S. Nakamoto, ‘Bitcoin: A Peer-to-Peer Electronic Cash System’, *Satoshi Nakamoto Institute*, 2008, [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [32] R. Kannan, *FROST: Python Implementation Toolkit and Runtimes for FROST Signatures on Different Curves*. (2023). [Online]. Available: <https://github.com/Raadhesh/FROST>
- [33] Zcash Foundation, ‘FROST — Performance’. 2021. [Online]. Available: <https://github.com/ZcashFoundation/frost/blob/3ffc19d8f473d5bc4e07ed41bc884bdb42d6c29f/performance.md>
- [34] T.-S. Ng, S.-Y. Tan, and J.-J. Chin, ‘A variant of Schnorr signature scheme with tight security reduction’, in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, Oct. 2017, pp. 411–415. doi: 10.1109/ictc.2017.8191014.
- [35] T.-S. Ng, S.-Y. Tan, and J.-J. Chin, ‘A Variant of BLS Signature Scheme with Tight Security Reduction’, in *Mobile Networks and Management*, J. Hu, I. Khalil, Z. Tari, and S. Wen, Eds, Cham: Springer International Publishing, 2018, pp. 150–163.