CSE 599k Final Submission - FROST and ROAST

Sela Navot Gautham Anant

Introduction

The main goal of this project was to learn about the FROST protocol by reading [1] (excluding the BLS section) and [2], and to learn about ROAST by reading [3]. The outcome of this project is this report and implementation of FROST2.

Go Implementation of Frost2

Frost2 provides a nice scaling benefit over Frost1 in that it reduces the protocol to a constant number of group exponentiations for each signer, as opposed to linear in the number of signers. Such benefits are nontrivial in decentralized systems where the number of signers may be large. We have edited the canonical Go implementation of Frost1 (https://github.com/taurusgroup/frost-ed25519), which operates under a peer-to-peer setting between signers to create a completely coordinated (the DKG is still peer-to-peer) Frost2 implementation (github.com/gpsanant/frost-ed25519). Run

```
go test test/sign_test.go
```

to test it!

This Report

We were somewhat unsure of what to write in this report. Our project was about reading, and anything along the lines of copy-pasting/paraphrasing/summarizing the papers seemed a little pointless. Therefore, all we do in this report is give an informal high-level overview of some of the concepts in the papers we read. Additionally, in the Robustness and ROAST section, we included a few trivial things that were not included in the ROAST paper, with the goal of showing that ROAST is actually a pretty good scheme and, depending on the properties we want our scheme to satisfy, it might be impossible to do better.

We did not explain the FROST protocol, talk about many of the security proofs, nor about the attacks presented against FROST. We planned to, but suspect that the report is already too long as is.

Security Definitions

Defining the unforgeability of threshold signatures is harder than defining the security of normal signature schemes. [1] provides the following security definitions (stated informally here):

Consider a signature scheme Σ, a threshold t, and a set of signers S with $|S| \geqslant t$. Let $\mathcal{A}$ be an adversary who has full control over $c < t$ signers of the adversaries choice, chosen before the initiation of the protocol, and learn their private key. The adversary can ask honest signers for pre-signature shares and signature shares for adaptively chosen leader requests (in FROST, a leader request includes a message, a set of signers, and the pre-signature shares assigned to each signer).[1]. The winning condition is the following for each security definition (from strongest to weakest): The adversary wins if they can construct a valid (message, threshold signature) $= (m, \sigma)$ pair such that:

[1] this is equivalent to the adversary having corrupted the aggregator

- TS-UF-4: Adversary hasn't created a valid leader request with message m and queried a signature share from all honest servers in the signing set of the leader request.

- TS-UF-3: Adversary hasn't created a leader request with message m and queried a signature from all honest servers that have a valid pre-signature share in the leader request.

  Note that if a scheme is TS-UF-3 secure, it is easy to make it TS-UF-4 secure at the cost of additional work for the signers: we can have each signer authenticates their pre-signature share (can be done via a digital signature), include the authentications in the leader request, and have the honest signers only provide partial signatures for leader requests with valid authentications. Informally, this would mean that an adversary cannot gain information or forge a signature by injecting incorrect or malicious pre-signature shares in the name of honest servers into leader requests, thus bridging the difference between TS-UF-3 and TS-UF-4.

- TS-UF-2: Adversary hasn't created a leader request with message m and queried signatures shares from $t - c$ honest signers on this leader request.

- TS-UF-1: The adversary hasn't requested a total of $t - c$ signature shares across all leader requests with the message m.

- TS-UF-0: The adversary hasn't requested a single signature share for a leader request with message m.

  Simply satisfying the definition of TS-UF-0 is not enough in order for a threshold signature scheme to be secure, since even if the scheme is TS-UF-0 secure it might be possible for the adversary to create a threshold signature for a message that less than t of the signers wanted to sign (the minimum number of signers who need to sign the message is $c + 1$: the number of corrupt signers plus the one honest signer who provided a partial signature for this message, and $c + 1$ may be less than t).

  However, note that in the game the adversary is allowed to corrupt $t-1$ servers. Intuitively, if the adversary can break TS-UF-1 by corrupting $c < t - 1$ signers and querying partial signature from at most $|S| - c - 1$ honest signers, then the adversary can corrupt the honest servers instead of querying their signatures and break TS-UF-0 security. Thus, it is not surprising that TS-UF-0 implies TS-UF-1 (if we consider t and $|S|$ as constants and not security parameters).

  The only challenge is that the adversary must choose the corrupt adversaries in advance (possibly before it knows which partial signatures it wants to query). Thus, the reduction in [1] involves the adversary guessing which of the $|S| - c - 1$ of the non-corrupt servers it wants to corrupt (i.e. which ones it would want to query if it was to try to break UF-TS-1 security), and therefore the success probability of the reduction algorithm might be a factor of $\frac{1}{\binom{|S|-c}{t-c-1}}$ smaller then the success probability of the original adversary (the probability that the adversary guesses right times the original success probability).

The security definitions above can also be extended to define strong unforgeability: a scheme that is TS-UF-i secure is strong TS-UF-i secure (TS-SUF-i) if it has a verification algorithm SVf that satisfies the following: for every signing session verification key VK and leader request LR, there exist at most 1 signature σ such that SVf(σ) returns true. In other words, for a given group public key there are no two different correct signatures for the same leader request.[2]

Security Definitions and Robustness

The major advantage of threshold signatures (over a group signature that requires all group members to sign) is that threshold signatures allow for robustness: if a small subset of the signers crash/are offline/retired/are malicious, the group can still produce signatures. Thus, we may want a scheme where we can use a leader request to produce a valid signature if at least t members of its signing set provide a signature share.

[2] I am a little confused on why we need the uniqueness of a verifiable signature in this definition. Isn't it enough to require that it is computationally hard for an adversary to find a different signature that verifies correctly for the same leader request? This seems more in line with the definition of strong unforgeability for regular signatures, as given in [4], for example

However, TS-UF-3 and TS-UF-4 schemes do not allow an honest coordinator to produce a signature unless they have a signature share from all members of a leader requests signing set. Indeed, FROST is therefore not robust (in fact even the variants that are not TS-UF-3 are not robust).

ROAST gets around this issue by allowing the aggregator to send out many leader requests, and therefore allows us to create a robust TS-UF-3 or TS-UF-4 scheme (more on this later). However, if we wanted the "perfect scheme:" a scheme that is robust and only requires the signers to sign a single leader request, it could not achieve more than TS-UF-2 unforgeability.

FROST 2 Security

We will provide an overview of the proof that FROST-2 is TS-SUF-2 secure under the OMDL assumption in the Random Oracle Model.

What does that mean?

The One More Discrete Logarithm (OMDL) assumption essentially states that an adversary who queries an oracle for n random group elements (i.e. challenges) and another oracle for the discrete logarithm of any group element at most $n-1$ times cannot provide the discrete logarithm of all n challenges. In essence, they cannot provide one more discrete logarithm than the number of discrete logarithms they queried.

TS-SUF-2 security is a specific type of unforgeability security which states for a signature on a message to be considered "forged", less than a threshold number of signers must have been sent the leader request that included the message (if there are corrupted signers, then less than a threshold minus the number of corrupted parties).

The proof shows that a simulating wrapper that outputs a solution to the OMDL game can be built around a forging FROST-2 adversary. The wrapper queries the OMDL challenge oracle for t group elements that it treats as coefficients of a polynomial from which it simulates the public keys of the signers. It queries the challenge oracle for a number of group elements to simulate the many nonces used in the frost protocol.

The crux of the proof relies on the fact that the execution of this wrapper can be "forked" just before simulating the leader request on the forged message and still succeed with a non-negligible probability. The forking leads to two different forgeries on the same message which can be used to reveal the threshold private key of the signing set without a query to the OMDL oracle. The wrapper can also use the forked execution to find the private keys of all of the signers who responded to the forgery's leader request. One can show that, given the threshold private key, the private keys of signers who signed on both forks, the private keys of corrupted signers (simulated via a query to the OMDL discrete logarithm oracle), and the private key of any other signers (to get the total number of known signer private keys to be t-1), can find the discrete logarithm of all t group elements used the simulate the signer public keys. Again, this is enabled due to the fact the wrapper gets the threshold private key for free because the FROST-2 forging adversary makes no queries to the discrete logarithm oracle.

It can also be shown that, in order to solve the discrete logarithm challenges for all of the nonces, the wrapper ends up making a number of queries to the discrete log oracle that is equivalent to the number of nonce challenges. Remember, in FROST, nonces come in pairs. Nonce pairs that are not used have their discrete log queried directly to the oracle. Nonce pairs that are used already have a linear combination of their discrete logs queried to the oracle to simulate signatures and just need a query on one of them in order to calculate the discrete log of the other.

So overall, we see that the wrapper can provide the discrete log of all of the nonces and coefficient group elements in the threshold polynomial while making a lesser number of queries to the discrete log oracle, winning the OMDL game. Since we assume that OMDL is hard, creating an adversary that breaks the TS-SUF-2 security of FROST2 is hard.

Robustness and ROAST

ROAST: Introduction

In [3] Ruffling et al. introduced a wrapper method ROAST(Σ) that uses a threshold signature scheme with identifiable aborts Σ to create a robust threshold signature scheme. ROAST guarantees that if a signing session of N signers has a non-malicious coordinator[3] and at least t of the signers wish to sign the message then the session produces a valid threshold signature, even if any subset of the servers who do not wish the message to be signed are malicious and attempt to hinder the protocol.

As in the ROAST paper, we will mostly consider Σ being one of the variants of FROST, which are unforgeable, provide identifiable aborts, but are not robust: even a single malicious or unresponsive server in a FROST signing session causes the session to fail. In particular, if a server in a FROST signing session goes offline or is prohibitively slow to respond, the FROST protocol will not terminate or be forced to terminate prematurely without producing a valid threshold signature. Hence, using ROAST(FROST) instead of FROST provides a non-trivial robustness guarantee.

However, there is a cost: servers in the ROAST signing session may have to provide a signature share for multiple leader requests ($N-t+1$ such requests in the worst case). Below, we will show that this cost is inherent to robust schemes that are TS-UF-3 secure. I.e. there does not exist a scheme that is perfectly robust, TS-UF-3 secure, and the coordinator issues less than $N-t+1$ partial signatures per signing session in the worst-case scenario.

We will also show that ROAST is as unforgeable as the underlying threshold signature scheme. Consequently, using ROAST with FROST1 as the underlying scheme provides a robust and TS-UF-3 secure scheme (under the OMDL assumption and in the ROM) [1]. Furthermore, [1] provides a construction of a TS-UF-4 secure threshold signature scheme using FROST1 and a digital signature scheme. Using this scheme as the basis for ROAST, we get a robust and TS-UF-4 secure threshold signature scheme (again, under the OMDL assumption and in the ROM).

Identifiable Abort

ROAST requires the underlying signature scheme to have identifiable aborts. In [3], the definition of identifiable aborts is that in a failed signing session, the coordinator can identify one signer as malicious, and that signer is one of the signers responsible for the failure. They also provide a game definition for this property (based on previous work such as [5]) as well as a proof that their variation of FROST indeed provides identifiable abort.

However, it seems that variations of FROST satisfy a stronger property: if a signing session fails, the coordinator can distinguish all the signers responsible for the failure from the honest signers. We will not prove it, and would not be surprised if it is already proven, but we will refer to this property as **strong identifiable abort**.

ROAST works more efficiently with an underlying scheme that satisfies the strong identifiable abort property, and it might have made a difference in our result if we were looking at probabilistic robustness as opposed to perfect robustness (a protocol success probability of 1).

ROAST: The Scheme

Let $\Sigma = \{$KG, SPP, LPP, LR, PS, Agg, Vf, PVf$\}$ be a threshold signature scheme with identifiable abort that consists of:

- Σ.KG: the distributed key generation algorithm of Σ. At the completion each signer i has a private key s_i, and a group verification key VK is produced. In the case of FROST, each signer also gets a public key VK_i.
- Σ.SPP and Σ.LPP refer to the coordinator and server pre-processing algorithms respectively. In particular, these algorithm are not dependent on the message to be signed

[3] good behavior of the coordinator is required for robustness, but not unforgeability

- In Σ.LR refers to the production of a leader request by the coordinator request
- Σ.PS is the partial signature algorithm each server runs, and is dependent on the message and the server secret key
- Σ.Agg is used by the server to coordinator to aggregate partial signatures into a threshold signature.
- Σ.Vf is a verification algorithm that verifies whether a threshold signature is valid
- Σ.PVf is a partial verification algorithm, and is the algorithm that distinguishes a threshold signature scheme with strong identifiable aborts. This algorithm allows the coordinator to check whether a partial signature provided by a signer is a valid signature share for a leader request.

Below we describe (somewhat informally) the algorithm of the coordinator role in ROAST. Note that we do not know how long signers take to respond to requests, and thus the need for the "Upon [Condition]" parts of the algorithm which should be executed whenever the condition is met, or the condition "happens". We also omit the key generation and signers side of the protocol, since these simply use the appropriate algorithm provided by Σ.

The only significant difference between the psuedo-code provided below and the psuedocode provided in [3] is that the psuedocode provided below allows the coordinator to keep track of more than one pre-signature share from each user (as opposed to one in the original paper). This allows the coordinator to take full advantage of the semi-interactive nature of Σ: the signing server can provide the coordinator with pre-signature shares for multiple ROAST sessions in advance, allowing them to use the semi-interactive property of Ω.

How Efficient is ROAST

If all servers in the signing session are honest and responsive, then ROAST succeeds as soon as t signers produced a valid signature. Thus, in every signing session where FROST would have successfully produced a threshold signature, ROAST(FROST) succeeds just as efficiently. ROAST(FROST) is only inefficient compared to FROST in the case that there are malicious/unresponsive servers.

Furthermore, as we explain below, there does not exist a robust threshold signature scheme that is TS-UF-3 or TS-UF-4 secure and the leader issues less than $N-t+1$ leader requests in every signing protocol that includes at least t honest signers. Thus, we can not design a TS-UF-3 or TS-UF-4 secure scheme that is perfectly robust (i.e. success probability is 1)[4] and more efficient than ROAST in terms of how many leader requests the coordinator sends out.

In our model, an adversary initiates a threshold signing protocol with N signers and a threshold t. It then chooses a set C of signing servers to corrupt with $|C| \leqslant N-t$. The adversary can see the private state of each corrupted server and control if, when, and what they respond to partial signature requests. The adversary wins if it prevents the signing protocol from producing a valid signature. Note that the adversary cannot corrupt the aggregator.[5]

For completion, it should be noted that ROAST is robust against even stronger adversaries who may decide which servers to corrupt during the execution of the protocol.

Attack Against More Efficient TS-UF-3 Schemes

Let Ω be a robust threshold signature scheme that is TS-UF-3 and requires the leader to issue at most $N-t$ different leader requests. We will show that Ω is not perfectly robust (there exists an adversary that breaks the robustness of Ω with non-zero probability).

We will consider the two following cases saperately:

[4] I think everything we say would also work when actually considering success probabilities and computationally bounded adversaries, but it will take more work.

[5] I don't think it would be hard to write a game definition for this adversary

Algorithm 1 ROAST_COORDINATOR(Σ):

```
1: procedure INIT SESSION(SS, m)                ▷ SS: the set of signers, m: the message to be signed
2:     Save SS and m
3:     M ← ∅                                    ▷ The set of servers known to be malicious
4:     R ← ∅                                    ▷ The set of servers for which we have a pre-signature share
5:     P ←Array(|SS|)                           ▷ The presignature shares for each signer
6:     C ←Array(|SS|)                           ▷ which session each signer is currently assigned
7:     L ← ∅                                    ▷ Set of leader requests created
8:     for all s_i ∈ SS do
9:         P[i] ← {pre-signatures already provided by s_i}
10:        if P[i] ≠ ∅ then
11:            R ← R ∪ {s_i}
12:        else
13:            Ask s_i for a pre-signature share on LR
14:        end if
15:    end for
16: end procedure
17:
18:
19: procedure UPON [|R| ⩾ t]
20:    Choose S ⊆ R with |S| = t
21:    R ← R\S
22:    LR ← Σ.LR(m, S, P)
23:                                  ▷ Create leader request, assume it removes used pre-signatures shares from P
24:    L ← L ∪ {LR}
25:    LR.Signers← ∅
26:    for all s_i ∈ S do
27:        Send s_i a request to sign LR
28:        C[i] ← LR
29:    end for
30: end procedure
31:
32:
33: procedure UPON [RECEIVE SIGNATURE SHARE FROM s_i] (s_i, pσ)
34:    if ¬Σ.PVf(C[i], σ_i) then                ▷ If the partial signature is invalid
35:        M ← M ∪ {s_i}
36:    else
37:        LR.Signers← LR.Signers∪{(s_i, pσ)}
38:        C[i] ←null
39:        if P[i] ≠ ∅ then
40:            R ← R ∪ {s_i}
41:        end if
42:    end if
43: end procedure
44:
45:
```

46: **procedure** UPON [RECEIVE PRE-SIGNATURE SHARE FROM s_i] (s_i, ρ)
47: $P[i] \leftarrow P[i] \cup \{\rho\}$
48: **if** $C[i] \in$ {null, uninitialized} and $s_i \notin M$ **then**
49: $R \leftarrow R \cup \{s_i\}$
50: **end if**
51: **end procedure**
52:
53:
54: **procedure** UPON [$\exists s_i \in SS \backslash M : P[s_i] = \varnothing$] ($s_i$)
55: Request new pre-signature shares from s_i
56: **end procedure**
57:
58:
59: **procedure** UPON [$\exists LR \in L : |LR.\text{SIGNERS}| \geqslant t$] ($LR$, LR.Signers)
60: ▷ enough partial signatures!
61: **return** Σ.Agg(LR, LR.Signers)
62: **end procedure**
63:
64:
65: **procedure** UPON [$|M| > |SS| - t$]
66: ▷ Too many malicious signers
67: **return** FAIL
68: **end procedure**

1. The leader request of Ω exposes which signers are required to sign this leader request (i.e. a leader request LR exposes $LR.SS$). Note that this is the case when $\Omega =$ ROAST(FROST).
2. The leader request does not reveal this information.[6] Note that any attack against this case is clearly an attack against the first case, but the attack we present against this case has a much lower success probability.

Case 1. Consider an adversary $\mathcal{A}$ who chooses $N - t$ of the N signers to corrupt uniformly at random. The corrupted signers behave honestly during key generation and respond to pre-signature share requests as expected (i.e. honestly).

Whenever a leader request is sent to one of the corrupt signers, they meet up and choose together uniformly at random a single corrupted signer c that is part of this leader request. The selected signer c makes sure never to respond to this leader request, but the rest of the corrupted signers will respond to it as if they were honest.

Success Probability: Since the scheme is TS-UF-3 secure, and all servers provide valid pre-signature shares, a leader request cannot be used to create a threshold signature unless every signer in signing set of this leader request honestly signed this message (again, ignoring negligible probabilities). Every leader request that had a corrupt signer in its set of signers will have a signer that refuses to respond to the request, and therefore cannot be used to produce a threshold signature. Since the corrupt signers were chosen uniformly at random, the coordinator cannot distinguish them from the honest signers unless a corrupt signer behaves maliciously in the protocol - and at most one corrupt server acts maliciously for each leader request created.

For any signer s, let Hon(s) denote the event that s is honest. Let H denote the set of signers s satisfying $\mathbb{P}(\text{Hon}(s) > 0)$. Initially, $\mathbb{P}(\text{Hon}(s)) = \frac{t}{N}$ for each signing server s, and H is the set of all signing server. Note

[6] It would be interesting to construct a variation of FROST that hides the set of signing servers in each leader request, though hiding this information from servers in the leader request might undermine the reason we want TS-UF-3 and TS-UF-4 security in the first place.

that whenever $|H| > t$, the aggregator does not have a strategy to select the set of all honest servers with probability 1.

Following each leader request, the aggregator either wins (creates a valid threshold signature) or learns that one server is dishonest, in which case the cardinality of H decreases by one.[7] Thus, if the aggregator has not won after issuing ℓ leader requests, as long as $N - \ell > t$ the aggregator does not have a strategy to create a valid signature with probability 1. Thus, until request $N - t + 1$ in which $|H| = t$ when the aggregator select servers, the aggregator does not have a winning strategy of probability 1. Thus, if the aggregator is only allowed to issue $N - t$ leader requests, it cannot win with probability 1 and therefore the adversary wins with a non-zero probability.

Case 2. In this case, consider an adversary that chooses $N - t$ servers to corrupt uniformly at random. Whenever a corrupt server receives a leader request to sign, they toss a fair coin and respond with a valid signature share if the coin lands head, and not respond if the coin lands tail.

Success Probability: If the adversary is lucky and for each leader requests exactly one of the servers that are issued this request tosses a tail, then the success probability of this adversary is the same as the success probability of the adversary from case 1. Since the probability of tossing the lucky sequence is non-zero, and the adversary from case 1 has a nonzero success probability, this adversary has a non-zero probability of winning.

The Unforgeability of ROAST

It is easy to see that ROAST is exactly as unforgeable as the underlying threshold signature scheme.

Intuitively, if adversary could break the security of a threshold scheme Ω they can produce a non-trivial forgery of an Ω signature, which is also a forgery of a ROAST(Ω) signature. Conversely, if ROAST(Ω) was less secure than Ω then any adversary against Ω could run the ROAST(Σ) protocol since all the information it needs in order to simulate running ROAST(Ω) can be obtained by at most **ns**(**ns**$-t$) calls to the PPO and PSignO oracles.

Another way to look at it is that an adversary playing the game $G_{\mathsf{TS}}^{\mathsf{ts\text{-}uf\text{-}i}}$ (for $i \in \{0, 1, 2, 3, 4\}$) to break ROAST(Ω) would be given exactly the same oracles to access as if they would have played the same game to break Ω. Since a ROAST signature is just an Ω signature, the winning condition will also be the same. Thus, ROAST(Ω) is just as secure as Ω itself.

Since [1] presents a TS-UF-4 secure scheme with identifiable aborts, using ROAST on that scheme gives us a TS-UF-4 secure and perfectly robust scheme.

[7] the probability that each server is honest changes, but the only one that drops to zero is the probability that the server who revealed itself as malicious is honest

References

[1] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST. *Cryptology ePrint Archive*, 2022(833), 2022.

[2] Chelsea Komlo and Ian Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. *Cryptology ePrint Archive*, 2020(852), 2020.

[3] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust Asynchronous Schnorr Threshold Signatures. *Cryptology ePrint Archive*, 2022(550), 2022.

[4] Dan Boneh, Emily Shen, and Brent Waters. Strongly Unforgeable Signatures Based on Computational Diffie-Hellman. *Lecture Notes in Computer Science*, 3958, 2006.

[5] Rosario Gennaro and Steven Goldfeder. One Round Threshold ECDSA with Identifiable Abort. *Cryptology ePrint Archive*, 2020(540), 2020.