ATSSIA: Asynchronous Truly-Threshold Schnorr Signing for Inconsistent Availability

Snehil Joshi[1], Durgesh Pandey[1], and Kannan Srinathan[2]

[1] International Institute of Information Technology, Hyderabad - Telangana, India
{snehil.joshi,durgesh.pandey}@research.iiit.ac.in
[2] International Institute of Information Technology, Hyderabad - Telangana, India
srinathan@iiit.ac.in

Abstract. Threshold signature schemes allow any qualified subset of participants (*t-out-of-n*) to combine its shares and generate a signature that can be verified using a single threshold public key. While there are several existing threshold signature schemes, most are either *n-out-of-n* and/or require *consistent availability* of the exact same set of participants through several rounds. This can result in signer availability becoming a bottleneck in the signing process. Our threshold signature scheme removes this dependence by introducing truly threshold asynchronous signatures, i.e., once the message to be signed has been revealed, the signers simply sign and broadcast their signature. Our scheme also uses misbehaviour detection to impose accountability for invalid signing. We prove that our scheme is safe against known distributed attacks and is $EUF - CMA$ secure in the Random Oracle Model for up to $t - 1$ malicious participants.

Keywords: Digital Signature · Threshold Signatures · Secret sharing · Blockchain · Distributed protocols

1 Introduction

Currently, multi-signatures [12] [20] , aggregate signatures [13] and threshold signatures [12] are relevant approaches that are aiming to solve the problem of distributed signing. The *t-out-of-n* (t, n) threshold signature is a protocol which allows a subset S of the players to combine their shares and reconstruct the private key to sign a message if $, S, >= t$, but disallows generating a valid signature if $, S, < t$. This property holds even in the presence of colluding, malicious participants as long as they number less than the threshold t.
Threshold signatures have seen an increased interest since the advent of blockchain technology after Nakamoto's white-paper on Bitcoin [1] as these signatures can be used by participants for signing transactions in blockchains. However, with a *dynamic and distributed* system spanning the world, the problem of latency due to communication overhead or unavailability of participating nodes arises. Any improvement in this regard is very useful in not just blockchains, but any application requiring distributed permissions.

1.1 Related Work and Motivation

Several distributed signature schemes exist for DSA (and ECDSA), Schnorr and BLS signatures. One of the first ideas came from the paper by Micali et al [22] on accountable subgroup multi-signatures. Their protocol uses Schnorr signatures and takes three rounds to sign. First the validity of the individual signatures is checked and then they are counted to see if they reach the threshold value. Other multi-signature schemes use similar methods where each signature needs to be verified separately. Similarly for simple aggregate signatures, a *t-out-of-n* multi-signature would result in ${}^nC_t$ possible key pairs for the group signature. A neater solution would have a single public key that the aggregated threshold signature can be verified with.
In this direction, Shoup [5] developed a threshold scheme based on RSA signatures. Boneh et al [13,14] created the BLS signature scheme that uses pairing based cryptography to provide a more efficient signing. Similarly, Maxwell et al [12] used Schnorr aggregate signatures for signing blockchain transactions, Lindell et al [9,10] presented a result for threshold ECDSA signatures using multiplicative-to-additive technique while Gennaro et al [7,8] created a scheme for ECDSA signatures.
While current threshold signature work well in specific scenarios, they suffer from one or more of the following issues:

- For truly-threshold (t, n) non-interactive protocols, only pairing-based solutions [13] exist. Pairing-based protocols while being excellent, may be difficult to adopt in some systems due to compatibility issue with already deployed signing protocols as well as reliance on a different security assumption, namely pairings, which is an additional security assumption which everyone may not be willing to incorporate in their systems right away.
- For non-pairing based protocols, most protocols focus on the all-or-nothing case of $t = n$ [8,12,16] but ignore an efficient implementation of the same in *truly* threshold protocols where $t \leq n$.
- Even with the case of $t = n$, multiple interactive rounds are required in the signing phase (at least 2 for Schnorr signatures) [12] [16] in most cases to generate the group's threshold signature.
- A signer that opts to participate in the a Schnorr-based multi-party signing protocol has to be available in *every* round till the aggregation of the protocol. Even with more efficient approaches like Komolo and Goldberg [15], the signers once fixed in the nonce-determining round can not be substituted during the signing phase without invalidating the threshold signature.

Our work looks at all these issues and aims to improve upon them.

We take time here to compare our scheme with that of Komlo and Goldberg [15] since ours is most similar to it. There are two main advantages of our scheme over [15]:

- In [15], signature scheme a signature aggregator selects the participants for nonce-generation. Once this selection has been made, the exact same group of

signers need to sign the message in the singing round. This effectively makes the signing round all-or-nothing, where if a participant from the previous round is missing, it will result in an incomplete signature. In our scheme, the signing is truly independent in terms for choice of participants. When any $\geq t$ signers sign the message, it will always result in the correct threshold signature.
– The second advantage is in terms of rounds used. Our protocol does not require the signers to be online together at all in the signing phase. In [15], there are one round (with pre-processing) and two rounds (without pre-processing).

To obtain these advantages, we have to bear a overhead of communication complexity in our pre-processing round (an extra overhead of 2π distributed nonce-generations) as compared to simple commitments used in [15]. However, we consider it a fair trade-off in return of an asynchronous and robust signing round.

1.2 Contributions

We present an efficient threshold signing scheme based on Schnorr signature, ATSSIA, with a non-interactive signing phase allowing for players to asynchronously participate without requiring to be online simultaneously. It achieves the same level of security as single-Schnorr signatures while having all the following desirable properties:

– Truly (t, n) threshold
– Based on widely-used cryptographic assumptions (DLOG)
– Asynchronous non-interactive signing phase
– No presumption on choice of t participants that will sign
– No wait-latency for unavailable participants because of 4 above
– Secure against ROS Solver and Drijvers' attacks during concurrent signing
– Provides in-built misbehaviour detection

Table 1 illustrates these properties

Table 1. Comparison with other Threshold schemes

Scheme	Musig	Musig2	BLS	FROST (1R)	FROST (2R)	**ATSSIA**
truly (t,n)	No	No	**Yes**	**Yes**	**Yes**	**Yes**
dynamic participants	No	No	**Yes**	No	No	**Yes**
non-interactive	No (3)	No (2)	**Yes**	No (1)	No (2)	**Yes**
base scheme	**Schnorr**	**Schnorr**	Pairings	**Schnorr**	**Schnorr**	**Schnorr**
without pre-processing	**Yes**	No	**Yes**	No	**Yes**	No
robust	**Yes**	**Yes**	**Yes**	No	No	**Yes**

In addition, our scheme retains all the important properties of threshold signatures while having a single public key for the threshold signature [12]. All

the computations are distributed and the secret values are never directly reconstructed to avoid a single point of failure. Unlike Stinson and Strobl [30], our scheme supports asynchronous signing and is also secure against the ROS Solver [17] and Drijvers' attacks [29] under the Random Oracle Model. Additionally, we provide for misbehaviour detection for robustness. A faulty signature will be verified for misbehaviour and removed (and economically penalised in a blockchain setting) and the threshold signature can then be calculated as long as honest partial-signatures meet the threshold criteria.

1.3 Organization of the Paper

Our paper is divided into six sections. Section 1 introduces our main idea and underlines the motivation for our work. Section 2 consists of the preliminaries and definitions we will be using for the rest of the paper. Section 3 and 4 describe the protocol and the security proof for it respectively. The last section underlines the importance of the usage of our protocol in blockchain and upcoming directions in the research.

2 Preliminaries

2.1 Communication and Adversary Model

Our communication model has a reliable broadcast channel as well as *peer-to-peer* channels between all participating nodes. We assume a probabilistic polynomial time malicious adversary who can statically corrupt up to $t-1$ participating nodes. The key and nonce generation phases require *all* participants to be present. However, the signing phase doesn't require all the signing parties to be present at the same time. We also assume possibility of a rushing adversary.

2.2 Message Indexing

Since our protocol uses batch processing for generating pre-nonce values, we need a method for keeping track of the message to be signed in any signing round so signers can provide the correct partial-signature for that message. A reliable message indexing system will make it possible for any available signer i to sign multiple messages without risk of wasting its nonce on an out-of-queue message. This can be achieved using a message server which indexes the messages in a queue: $m_1....m_j...m_\pi$. Any signer i can check the message index j to be signed and match it against the appropriate pre-nonce values.
Please note that this does not affect the security of our protocol. Since our protocol is secure in concurrent signing, even a malicious adversary controlling the message server and $t-1$ signers will not be able to forge the threshold signature. At its best, a malicious message server can generate invalid threshold signatures

using different messages for the same round. Faulty threshold signatures thus generated will be promptly invalidated and removed during signature aggregation.

2.3 Cryptographic Assumptions

The DLOG assumption: Let p be a prime whose size is linear in k. Given a generator g of a multiplicative group G of order p and $a \in Z_p^*$, the discrete logarithm or $DLog$ assumption suggests that $Pr[A_{DLog}(g, g^x) = x] \in (\kappa)$ for every polynomial-time adversary A_{DLog}.
Now we will define the concepts of VSS,Threshold signatures and Schnorr signature scheme that we will use to build our protocol.

2.4 Threshold (t,n) Secret Sharing

Given integers t, n where $t \leq n$, a (t, n) threshold secret sharing scheme is a protocol used by a dealer to share a secret s among a set of n nodes in such a way that any subset of $\geq t$ can efficiently construct the secret, while any subset of size $\leq t-1$ can not. [23].
To distribute the secret, the dealer first randomly selects $t-1$ values $a_1, .., a_{t-1}$, and then uses them as coefficients of polynomial $f(x) = \Sigma a_i \cdot x_i$. The secret is defined as $f(0) = s$.
The dealer then assigns indices i to each of the n nodes and gives them their secret share as $(i, f(i))$. The secret can then be reconstructed using Lagrange interpolation with any subset of size $\geq t$. Since a minimum of t points are needed to represent the polynomial, no subset of size less than t can find the secret [23].

2.5 Verifiable Threshold (t, n) Secret Sharing

To prevent malicious dealing by the trusted dealer, we utilize verifiable secret sharing (VSS) [24]. The (t, n) VSS scheme consists of a sharing and a reconstruction phase.
In the sharing phase, the dealer distributes the secret $s \in G$ among n nodes. At the end of this, each honest node holds a share s_i of the secret. In the reconstruction phase, each node broadcasts its secret share and a reconstruction function is applied in order to compute the secret $s = Reconstruct(s_0, ...s_n)$ or to output $\perp$ indicating that the dealer is malicious.

2.6 The Schnorr Signature Scheme

Let $\mathbb{G}$ be the elliptic curve group of prime order q with generator G. Let H be a cryptographic hash function mapping to Z_q^*. For a message m the Schnorr signature is calculated as follows:

- *Key Generation*
  Generate public-private key pair: (x,$P = x \cdot G$) where x is a random point in Z_q and G is a generator in $\mathbb{G}$ be the elliptic curve group of prime order q with generator G

- *Signing*
  Choose random nonce $k \in Z_q$; Compute public commitment $r = k \cdot G$
  Compute: $e = H(P, r, m)$;
  Compute: $s = k + e \cdot x$
  The signature is $\sigma = (r, s)$

- *Verification*:
  Given values m, P, σ
  extract r and s from σ and compute $e_v = H(P, r, m)$
  compute $r_v = s \cdot G - e_v \cdot P$
  if $r_v = r$ then verified successfully

While we are specifically using Schnorr signature scheme over elliptic curves in our paper, our protocol can directly translate to the discrete logarithm problem as we are following standard group operations.

2.7 Threshold Signatures Schemes

Threshold signature schemes utilize the (t, n) security property of threshold protocols in aggregate signature schemes. This allows signers to produce a group signature over a message m using their secret key shares such that the final signature is identical to a single party one. This allows signing without knowing the secret and also verification of the threshold aggregate signature is as simple as that of a regular signature. In threshold signature schemes, the secret key s is distributed among the n participants, while a single key P is used to represent the group's public key.
To avoid point of single failure, most threshold schemes also need to generate their shares distributively instead of relying on a trusted dealer [26].

3 ATSSIA: Asynchronous Truly Threshold Schnorr Signatures for Inconsistent Availability

3.1 Overview

A signature scheme has three stages: *KeyGen*, *Sign* and *Verify*. We modify it by adding a pre-processing phase for nonce-generation *PreNonceGen* and replace the *Sign* phase with two separate phases of *PartSign* and *Aggregate*. We also provide the optional phase for misbehaviour detection in case of faulty partial signatures being generated.

To keep our signing phase non-interactive, we generate pre-nonce values for all participants in the interactive *PreNonceGen* phase. All the n participants are indexed using numbers $\{1, ...n\}$ both for clarity as well as ease of calculating Lagrange coefficients and other values. The participant index values are made publicly available. Additionally, we have a message indexing system that maintains an index table for each message being broadcast for signing. This index is also publicly available. This is a necessity for coordination for concurrent message signing in a non-interactive signing phase as signers need to match the values produced in *PreNonceGen* phase with the corresponding message.
Our signature scheme is denoted by ($\mathbb{G}$,q,g), where q is a k-bit prime, $\mathbb{G}$ is a cyclic group of order q and G is the generator of $\mathbb{G}$. The correctness for group parameter generation follow standard procedures that can be verified.
We provide an overview of our scheme next.

Overview of ATSSIA

- **KeyGen** : All n-participants generate their key pairs (x_i, p_i) using a DKG

- **PreNonceGen**: All n-participants generate π pairs of pre-nonce pairs ${k_{ij}}^a$ and ${k_{ij}}^b$ using π parallel DKGs. They broadcast the corresponding commitment values of ${r_{ij}}^a = {k_{ij}}^a \cdot G$ and ${r_{ij}}^b = {k_{ij}}^b \cdot G$

- **PartSign**: When an individual signer receives a message m_j for it confirms the message index j and signs the message using the corresponding nonce value:
  $\sigma_{ij} = (s_{ij}, R_j, r_{ij})$ where:
  $s_{ij} = ({k_{ij}}^a + h_j \cdot {k_{ij}}^b) + H_m(m_j, R_j, P) \cdot x_i + \rho \cdot H_m(m_j, r_{ij}, p_i) \cdot x_i$
  (we will explain in Section 3.3 how the values h_j,R_j and ρ are calculated)
  This value is then broadcast for aggregation.
- **Aggregate**: Once at least t valid partial-signatures have been broadcast for a given message, they can be aggregated publicly by anyone using Lagrange interpolation
  $S_j = \sum_i \lambda_i \cdot s_{ij}$

- **Verify**: The final signature can be verified as:
  $(S_j \cdot G)$ *mod* $\rho =? R_j + H_m(m_j, R_j, P) \cdot P$
  (mod by ρ removes the misbehaviour detection portion of the signature)

- *Misbehaviour Detection (Optional)*: A misbehaving signer can be detected by verifying the individual signer's signature using:
  $S_j \cdot G =? R_j + H_m(m_j, R_j, P) \cdot P + \rho \cdot H_m(m_j, r_{ij}, p_i)) \cdot p_i$

3.2 KeyGen

We use a variation of Pedersen's DKG scheme [18] by Komlo and Goldberg [15] to generate our threshold keys. It differs from Pedersen's scheme by providing security against a rogue-key attack for a dishonest majority setting by demanding a ZKPoK from each participant w.r.t their secret. Note that Pedersen's DKG is simply where each participant executes Feldman's VSS as the dealer in parallel, and derives their secret share as the sum of the shares received from each of the n VSS executions. So we get n eligible nodes running n VSS protocols in parallel to distributively generate the shares for the secret key X for the threshold signature. The 2-round KeyGen phase is as follows:

KeyGen

We assume *WLOG* the n participants are indexed as i where $i \in 1, 2...n$. This improves reading the text of the protocol and makes it easier to calculate their Lagrange coefficients in the aggregate phase.

1. Each participant i generates t random values $(a_{i_0}, ..., a_{i_{t-1}})$ in Z_q and uses these values as coefficients to define a polynomial $f_i(x) = \Sigma_{l=0}^{t-1}(a_{i_l} \cdot y^l)$ of degree $t-1$ in Z_q

2. Each i also computes a proof of knowledge for each secret a_{i_0} by calculating $\sigma = (w_i, c_i)$ where a_{i_0} is the secret key, such that $k \in Z_q, R_i = k \cdot G, h_i = H(i, S, a_{i_0} \cdot G, R_i), c_i = k + a_{i_0} \cdot h_i$, where S is the context string to prevent replay attacks.

3. Each i then computes its public commitment $C_i =< A_{i_0}, ..., A_{i_{t-1}} >$, where $A_{i_j} = a_{i_j} \cdot G$, and broadcasts (C_i, σ_i) to all other participants.

4. After participant i receives (C_j, σ_j), $j \neq i$, from all other participants $j \neq i$ it verifies $\sigma_j = (w_j, c_j)$ by computing $h_j = H(j, S, A_{j_0}, w_j \cdot G)$ and then checking for $w_j = c_j \cdot G - A_{j_0} \cdot h_j$ with $\perp$ (abort) on failure.

5. Next each i sends the secret share $(i, f_i(j))$, to every other participant P_j while keeping $(i, f_i(i))$ for itself.

6. Now each i verifies its shares by checking if $f_j(i) \cdot G = \Sigma(A_{j_k} \cdot (i^k mod q)), k = 0...t-1$, with $\perp$ on failure.

7. Each i finally calculates its individual private key share by computing $x_i = \Sigma_j f_j(i), j = 1...n$, and stores $x_(i)$ securely.

8. Additionally, each i also calculates its public share for verification $p_i = x_(i) \cdot G$, and the group's public key $P = \Sigma_j^n A_{j_0}, j = 1...n$. The participants can then compute the public share for verification for all the other participants by calculating $p_j = \Sigma_j^n \Sigma_k^{t-1}(A_{j_k} \cdot (i^k mod q)), j = 1...n, k = 0...t-1$). These p_i values also act as the individual public key for the corresponding i participant.

At the end of the KeyGen phase we get:
partial private key shares: x_1 $...x_n$ partial public key shares: p_1 $...p_n$
group public key: P where $P = X \cdot G$
Each participant has the value (i, x_i). Any $\geq t$ such values can combine to give the secret X.

We point here that there might be more scalable protocols available with similar level of security. For example, Canny and Sorkin [27] can provide more efficient (poly-logarithmic) DKGs. However, all such protocols currently require special use-cases of low tolerance and a trusted dealer in the pre-processing phase which makes them impractical for our specific case of truly distributed threshold signing.

3.3 PreNonceGen

Before signing a message m for a round j, at least t nodes need to collaborate to generate a unique group nonce R_j. Since this can not be achieved using deterministic nonce generation in our case, we opt to generate and store nonces in a batch of pre-determined size ϕ in a pre-processing phase.

Gennaro [28] presented a threshold Schnorr signature protocol that uses DKG to generate multiple nonce values in a pre-processing stage independently of performing signing operations. We use the same approach to our problem but use the modified DKG as used in KeyGen. This leads to a communication overhead, but is still the most secure method to achieve a non-interactive signing phase.

In the **PreNonceGen** phase we do not use the DKG directly to generate individual nonces and commitments. This step is crucial to prevent two specific attacks used to forge signatures in a concurrent signing protocol.
In order to make our signing phase completely non-interactive, we need to ensure that no extra interaction rounds are needed while calculating the threshold nonce commitment values. However, as shown by Maxwell et al. [12], this opens the possibility of the ROS Solver [17] and Drijvers' attacks [29]. These attacks rely on the attacker's ability to control the signature hash by controlling the threshold nonce value R_j, by either adaptively selecting their own commitment after victim's nonce commitment values are known, or by adaptively choosing the message to be signed to manipulate the resulting challenge for the set of

participants performing the signing operation. The obvious way to avoid this, is by committing the r_ij values before the message is revealed so it can not be adaptively changed later.
However, since we want to prevent these attacks *without* introducing an extra round of interaction or sacrificing concurrency, we instead use a modification of the work by Nick et al [16]. Their approach essentially binds each participant's nonce commitment to a specific message as well as the commitments of the other participants involved in that particular signing operation.
To achieve this, our scheme replaces the single nonce commitment r_{ij} with a pair (r^a_{ij},r^b_{ij}). Each prospective signer then commits r^a_{ij} and r^b_{ij} in the pre-processing phase and given h_j is the output of a hash function $H_r()$ applied to all committed pre-nonces in the previous phase, the threshold public key, and the message m_j, the nonce commitment for any signer i will now be:
$r_{ij} = r^a_{ij} + h_j.r^b_{ij}$.
To initiate an attack, when any corrupt signer changes either its nonce values or the message, it will result in changing the value of h_j, thereby changing the nonce-commitment of honest signers as well. Without a constant nonce-commitment value, the attacks in [17] and [29] can't be applied. We can therefore overcome these two attacks without needed an extra commitment round.
So the **PreNonceGen** phase will use $2 * \phi$ parallel DKGs to generate batch of ϕ pre-nonce pairs per participant for a total of π prospective messages.

PreNonceGen

1. Each participant needs to conduct 2π parallel nonce-generation protocols to generate two shares (k^a_{ij}, k^b_{ij}) per nonce for the prospective j^{th} message.
   WLOG we will show how to distributively generate k^a_{ij} for all participants. The same procedure will apply to k^b_{ij}.
2. For every k^a_{ij} value, each participant i generates t random values $(a_{i_0}, ..., a_{i_{t-1}})$ in Z_q and uses these values as coefficients to define a polynomial $f_i(x) = \Sigma^{t-1}_{l=0}(a_{i_l} \cdot y^l)$ of degree $t-1$ in Z_q
3. Each i also computes a proof of knowledge for each secret a_{i_0} by calculating $\sigma = (w_i, c_i)$ where a_{i_0} is the secret key, such that $k \in Z_q, R_i = k \cdot G, h_i = H(i, S, a_{i_0} \cdot G, R_i), c_i = k + a_{i_0} \cdot h_i$, where S is the context string to prevent replay attacks.
4. Each i then computes its public commitment $C_i =< A_{i_0}, ..., A_{i_{t-1}} >$, where $A_{i_j} = a_{i_j} \cdot G$, and broadcasts (C_i, σ_i) to all other participants.
5. After participant i receives (C_j, σ_j), $j \neq i$, from all other participants $j \neq i$ it verifies $\sigma_j = (w_j, c_j)$ by computing $h_j = H(j, S, A_{j_0}, w_j \cdot G)$ and then checking for $w_j = c_j \cdot G - A_{j_0} \cdot h_j$ with $\perp$ (abort) on failure.
6. Each i next sends the secret share $(i, f_i(j))$, to every other participant P_j while keeping $(i, f_i(i))$ for itself.
7. Each i now verifies its shares by checking if $f_j(i) \cdot G = \Sigma(A_{j_k} \cdot (i^k mod q)), k = 0...t-1$, with $\perp$ on failure.

8. Each i finally calculates its individual pre-nonce share by computing $k_{ij}^a = \Sigma_j f_j(i), j = 1...n$, and stores k_{ij}^a securely.
9. Additionally, each i also calculates the corresponding public share for nonce verification $r_{ij}^a = k_{ij}^a \cdot G$
10. The participants can also compute the public share for verification for all the other participants by calculating $r_{ij}^a = \Sigma_j^n \Sigma_k^{t-1}(A_{j_k} \cdot (i^k mod q)), j = 1...n, k = 0...t-1)$

As is evident from above, the **PreNonceGen** phase of our protocol is the main bottleneck in terms of efficiency. All n participants are required to be present in this phase and each one will have to do 2π nonce-generations to prepare nonces for up to π prospective signatures. Unfortunately, so far, there has been no other way to remove or reduce this overhead and this is an unavoidable trade-off if we want to prevent the availability bottleneck in the signing round.
At the end of the **PreNonceGen** stage, every eligible signer i ends with a private nonce share of (k_{ij}^a, k_{ij}^b) to sign the prospective j^{th} message as well as the nonce commitment pairs of all other signers (r_{ij}^a, r_{ij}^b).
The values of private nonce shares and commitments are not yet determined. They will be generated when the message is known as we will explain in the **PartSign** stage.

3.4 PartSign

The signing phase of our protocol commences once the message to be signed is revealed. Before signing, the signer needs to determine the value of threshold nonce-commitment. Doing this *without* an extra round of commitment would normally make the scheme prone to the ROS Solver [17] and Drijvers' attacks [29] but we take care of that through the **PreNonceGen** phase.
Now we account for the values each participant i has received so far at the beginning of the **PartSign** phase. We denote the individual signer with its participant-index i and others with index o in order to avoid any confusion with the message index denoted by j.
private group key: X (VSS distributed) public group key: P
private key share: x_i public key shares: $p_o{=}x_o \cdot G$
Tuples of π individual pre-nonce commitment pairs: (r_{oj}^a, r_{oj}^b) for each participant i and prospective message m_j
From the values of (r_{oj}^a, r_{oj}^b) the signers can calculate the π threshold pre-nonce commitment pairs (R_j^a, R_j^b)
The VSS distributed values are not yet constructed and can only be calculated by Lagrange interpolation of at least t corresponding shares. Optionally, all the public/commitment values can be broadcast publicly or stored on a trusted server for use by an outside party for signature aggregation and verification.

In our scheme, the partial-signature also contains an additional portion with the signer's individual nonce-commitment as a challenge. This is done in order to make individual misbehaviour detection possible in case an invalid partial-signature is sent. This is explained in detail in *Section 3.7*. It doesn't affect the functionality or security of the rest of the protocol.

PartSign

For the j_{th} message $m_j \in Z_q$ and available hash functions $H_r(.)$ and $H_m(.)$ mapping $\{0,1\}^*$ to $\{0,1\}^l$ in Z_q^*, each individual signer calculates its partial-signature:

1. Each signer i for the message m_j sums the pre-nonce commitment pairs of all the eligible participants from *PreNonceGen* for the message m_j, (r_{oj}^a, r_{oj}^b) and individually calculates the value:
   $h_j = H_r(m_j, \sum_{o=1}^{n} r_{oj}^a, \sum_{o=1}^{n} r_{oj}^b, P)$

2. Each signer i calculates its own nonce value for m_j and the value of at least t (including self) nonce-commitments using:
   $k_{ij} = k_{ij}^a + h_j.k_{ij}^b$ *(for own nonce)*
   $r_{ij} = k_{ij} \cdot G$ *(for own nonce-commitment)*
   $r_{oj} = r_{oj}^a + h_j.r_{oj}^b$ *(for nonce-commits of other signers)*
3. Next the signer i uses any t values of r_{oj} calculated in step 2 and reconstructs the group nonce-commitment value by Lagrange interpolation as:
   $R_j = \sum_i \lambda_i.r_{oj}$ *(λ_i are the corresponding Lagrange coefficients)*
4. Now signer i can sign the message m_j as:
   $s_{ij} = k_{ij} + H(m_j, R_j, P) \cdot x_i + \rho \cdot H_m(m_j, r_{ij}, p_i) \cdot x_i$

   The signer i broadcasts $\sigma_{ij} = (s_{ij}, R_j, r_{ij})$

3.5 Aggregate

Aggregation of the threshold signature is rather simple and straightforward. For a given message m_j, once at least t partial-signatures have been broadcast, they can be aggregated using simple Lagrange interpolation
We can use a designated signature aggregator like Komlo and Goldberg [15], to improve efficiency of our **Verify** stage: signer sending single message to an aggregator vs broadcasting messages to everyone. As long as the aggregation is done by anyone in at least a semi-honest way, the threshold signature will be correctly constructed. The duty of the aggregator can be undertaken by one or more of the signers themselves, in which case, the signer that decides to be the aggregator will have to be present throughout the partial signing phase until the

threshold qualifying number of honest partial-signatures is not met.
Along with aggregating the threshold signature, an additional role of the aggregator is to ensure that the threshold signature is formed correctly by verifying it using the group public key. In case of a discrepancy, the aggregator should be able to detect the misbehaving signature shares from the signature, upon which it can remove those and use other available honest partial-signature(s) to re-aggregate the threshold signature. As long as t honest partial-signatures are available, the correct signature can be reconstructed.
Even a malicious aggregator can not forge the threshold signature or learn anything about the signing parties thereby maintaining EUF-CMA security. If it falsely accuses honest signers of misbehaviour or constructs an incorrect signature using fake shares, both those results can be later verified independently. A malicious aggregator can also deny constructing the correct signature. However in case of blockchains, semi-honest behaviour can be imposed on the aggregator via the incentive of a financial payment per honest aggregation and/or a deterrent in form of a financial penalty per wrongfully submitted signature since the final signature is publicly verifiable.
WLOG, for the rest of this paper, we will assume an external party as the aggregator.

Aggregate

1. The aggregator waits for the partial-signatures to be broadcast. As each $\sigma_{ij} = (s_{ij}, R_j, r_{ij})$ is received for some message m_j the aggregator checks it for validity using misbehaviour detection steps from *Section 3.7*.
   The partial-signatures that fail verification are discarded.
2. Once at least t correct partial-signatures are received, the aggregator generates Lagrange coefficients λ_i for each partial-signature using index values i of the corresponding signers.
3. The threshold signature can now be aggregated as:
   $S_j = \Sigma\lambda_{ij} \cdot s_{ij}$
   The aggregator broadcasts the threshold signature as $\sigma_j = (S_j, R_j)$

While this approach is sufficient in itself, in a special setup like a blockchain, we can leverage the environment to our advantage. In such a scenario, we can take a more optimistic approach by assuming that malicious behaviour is rare and instead of checking for partial-signature validity for each σ_{ij}, we can allow for misbehaviour to take place, as long as it can be detected and the faulty signer economically penalised. Misbehaviour will be easy to prove publicly later since all required values are public. If sufficient correct partial-signatures ($>= t$) are still available for a given message, the aggregator can replace erroneous partial-signatures with correct ones and re-aggregate the signature.

Assuming rational participants, the threat of a penalty should drastically reduce the probability of invalid partial-signatures. It will make this alternative approach more efficient as in most cases, only the threshold signature will be needed to be verified. We now illustrate this alternate approach.

Alternate Aggregate with Penalty

1. The aggregator waits for the partial-signatures to be broadcast. Once at least t correct partial-signatures $\sigma_{ij} = (s_{ij}, R_j, r_{ij}$ are received, the aggregator generates Lagrange coefficients λ_i for each partial-signature using index values i of the corresponding signers.
2. The threshold signature is aggregated as:
   $S_j = \Sigma\lambda_{ij} \cdot s_{ij}$
3. The aggregator now checks the validity of the threshold signature S_j as in *Section 3.6*:
   $(S_j \cdot G) mod \rho =? R_j + H_m(m_j, R_j, P) \cdot P$
4. If this verification succeeds, aggregator goes to step 6. If verification fails, the aggregator starts checking the individual partial-signatures σ_{ij} for misbehaviour using steps from *Section 3.7* and removes them. IT additionally penalises the signers that sent an invalid signature.
5. After all misbehaving partial-signatures have been removed, the threshold signature value S_j will be re-aggergated if at least t valid signatures are available:
   $S_j = \Sigma\lambda_{ij} \cdot s_{ij}$
6. The aggregator broadcasts the threshold signature as:
   $\sigma_j = (S_j, R_j)$

3.6 Verify

Verification of the threshold signature is the same as the standard Schnorr signature verification with a small modification: we mod the LHS of the verification equation with ρ before checking it. This is done in order to remove the individual signer's misbehaviour detection portion from the signature.
The new verification phase looks like:

Verify

Given values m_j, P, σ_j and ρ, the threshold signature is verified as follows:

1. Verifier parses σ_j to get S_j and R_j

2. Verifier then removes the misbehaviour detection portion from the signature using:
   $S = S_j mod \rho$
3. Next it calculates:
   $e_v = H_m(m_j, R_j, P)$
4. From e_v it calculates the expected nonce-commitment value:
   $R_v = S \cdot G - e_v \cdot P$
5. if $R_v = R_j$ then signature is valid, else invalid

3.7 Misbehaviour Detection

Among mutually distrusting, distributed parties, misbehaviour by one of the parties might warrant immediate detection and subsequent adjustment in the protocol. To achieve this, we generated an additional portion in our signature as the individual signers misbehaviour detection part. All the other phases of the signature scheme remained the same. The only change that occurred, was in the **PartSign** phase of the scheme. We elaborate on this.

Modified Partial-Signature Misbehaviour detection required the addition of an individual identifier to the regular signature. Our scheme achieved this by adding a new portion to the partial-signature (see *Section 3.4*) that contains the nonce-commitment for the individual signer i as part of the challenge value in the hash. Given the regular signature as:
$s_{ij} = k_{ij} + H(m_j, R_j, P) \cdot x_i$
The new signature became:
$s_{ij} = k_{ij} + H(m_j, R_j, P) \cdot x_i + \rho \cdot H_m(m_j, r_{ij}, p_i) \cdot x_i$
The signature value was modified to:
$\sigma_{ij} = (s_{ij}, R_j, r_{ij})$

Choosing the ρ value: The ρ value should have these features:

1. It should be sufficiently large so that modulus of the signature with ρ still preserves the rest of the signature as before, i.e., $\rho > (k_{ij} + H_m(.) \cdot x_i) \forall i, j$
2. It should be kept as small as possible to keep the signature footprint as close to the original partial-signature.

Combining these two properties we get ρ to be a prime number such that, $q+q^2 < \rho < q^3$ keeping it as close to $q^2 + q$ as possible. This makes our signature of a size around $2q$ to $3q$ which is similar to that of the original partial-signature. As we see in the next step, this does not affect the size of the aggregate threshold signature since the individual component will be removed before aggregation.

Verification of the threshold signature The new addition didn't change the way we aggregate the threshold signature. While aggregating we simply removed the individual identifier by taking the partial-signature modulus rho, i.e., $S = S_j mod \rho$ and followed regular verification from thereon (see *Section 3.6* for details).

Checking for misbehaviour Of our two approaches to signature aggregation, the first one necessarily while the second one in rare cases, requires misbehaviour detection to remove invalid signatures from the aggregation pool. The aggregator, verifier or any interested party can now check a given partial-signature for misbehaviour as shown below:

Misbehaviour Detection

Given values m_j, P, σ_{ij}, p_i and ρ, misbehaviour detection is tested as follows:

1. Verifier parses σ_{ij} to extract s_{ij}, R_j and r_{ij}
2. Verifier next calculates:
   $e_v = H_m(m_j, R_j, P) + \rho \cdot H_m(m_j, r_{ij}, p_i)$
3. From this e_v it calculates the expected nonce-commitment value:
   $r_v = s_{ij} \cdot G - e_v \cdot P$
4. if $r_v = r_{ij}$ then its a valid partial-signature, else misbehaviour has been detected

Cost analysis In this section, we provide the total cost in terms of round communication and exponentiation (the $\cdot G$) computation for participants in various phases of the protocol.
For **KeyGen** phase, a participant requires two round of communication : one broadcast round and another P2P communication round. A participant need to broadcast the proof of knowledge of a secret a_{i_0} along with commitment of t coefficient A_{i_j}. After each participant verify the correctness of broadcast data of other participants, a party need to send the polynomial f_i's evaluation for player j along with his own id. In complete key generation phase, a player need to compute a total of $(3n + t)$ $\cdot G$ computation: 2 computations in step 2 for calculating R_i and h_i, t computations for calculating commitment C_i in step 3, $2 \times (n - 1)$ for calculating w_j and h_j in step 4, $(n - 1)$ for calculating $f_j(i) \cdot G$ in step 6 and one computation in step 8 for calculating p_i.
The cost of generating pre-nonce values for a batch size of π by following is $2\pi\times$ that of the **KeyGen** phase as we need to run two DKG protocol for per participant to generate the two pre-nonces that will be used to calculate the nonce value later.
The **PartSign** doesn't involve any round of communication and is completely

non interactive. The partial signature just need to be broadcast or sent to aggregator once it is generated. It requires just one exponentiation computation for calculating r_{i_j} in step 2.
The simple **Aggregate** phase just waits for all the partial signatures to arrive and doesn't require any communication round or exponentiation computation.
The **Aggregate with Penalty** phase requires one computation in step 2, a variable number of misbehaviour detection cost and communication, spanning from 0 to $n - t$ depending on the number of misbehaving parties present.
The **Verify** phase just requires one exponentiation computation. The additional misbehaviour detection phase for detecting misbehaviour in one partial signature also requires only one such computation.

4 Proof of Security for ATSSIA

4.1 Proof of correctness

The various portions of the signature satisfy the properties of digital signatures. The key and nonce are distributively generated using existed secure methods from Gennaro's DKG [28] combined with a **PreNonceGen** phase to keep it safe even with concurrency.
We prove correctness by demonstrating how our scheme is equivalent to a single signer Schnorr scheme.

Theorem 1. *For some message m, given a public-private key pair (X, P), pre-nonce and pre-nonce commit values pairs (K^a, R^a) and (K^b, R^b) and $H_r$andH_m, the threshold signature generated using ATSSIA is equivalent to the single signer Schnorr signature:*
$S = K + e.X$
where $K = (K^a + H_r(m, K^a, K^b, P).K^b)$, $R = K \cdot G$ and $e = H_m(m, R, P)$

Proof. For our signature scheme, let there be some polynomials $f(.)$ and $(g_a(.), g_b(.))$ that are used to distribute the group key and pre-nonce values respectively, i.e., $f(.)$ distributes the private key X and $(g_a(.), g_b(.))$ distribute the pre-nonce pairs (K^a, K^b).
Therefore for a signer i we will get the signature value as $h(i)$:
$h(i) = (g_a(i) + c_r \cdot g_b(i)) + c_{m_R} \cdot f(i) + c_{m_{r_i}} \cdot f(i)$
where $c_r = H_r(m, \sum r_i^a, \sum r_i^b, P)$, $c_{m_R} = H_m(m, R, P)$ *and* $c_{m_{r_i}} = H_m(m, r_i, p_i)$
Lagrange interpolation on this results in:
$S = \sum(\lambda_i(g_a(i) + c_r \cdot g_b(i))) + (c_{m_R} + c_{m_{r_i}}) \cdot \sum(\lambda_i \cdot f(i))$
$S = K^a + c_r \cdot K^b + (c_{m_R} + c_{m_{r_i}}) \cdot X$
substituting notations by $K = K^a + c_r \cdot K^b$ and $e = (c_{m_R} + c_{m_{r_i}})$, we get:
$S = K + e \cdot X$, which is equivalent to the single signer Schnorr signature with key X, nonce K and challenge e.

Since our scheme replicates a single-party Schnorr proof substituting the threshold key and nonce values for individual ones, it provides correctness at the same level as the single signer protocol.

4.2 Proof of EUF-CMA Security

We will prove security against existential unforgeability in chosen message attacks ($EUF-CMA$) by demonstrating that the difficulty to forge our threshold signature by performing an adaptively chosen message attack in the Random Oracle (RO) model reduces to the difficulty of computing the discrete logarithm of an arbitrary challenge value ω in the same underlying group, so long as the adversary controls only up to $t-1$ participants.

Our proof uses these main algorithms:

- The Forger $\mathcal{F}$ which is the undermost-lying algorithm. We assume that $\mathcal{F}$ has the property of forging a signature in our threshold signature scheme with probability ϵ in time t. WLOG we assume that $\mathcal{F}$ also controls $t-1$ signers in our protocol and plays the role of the signature aggregator.
- A simulator $\mathcal{A}$ which manages the input and outputs for $\mathcal{F}$ and also simulates the honest participants and the RO queries.
- The Generalised Forking Algorithm $GF_{\mathcal{A}}$ which provides $\mathcal{A}$ with a random tape for its inputs and also provides outputs to its RO queries. It then utilises these to "fork" $\mathcal{A}$ to produce two forgeries $(\sigma, \sigma^{'})$ for the same RO query index.
- The Extractor algorithm $\mathcal{E}$ which takes the challenge value ω as input, embeds it into our scheme and then obtains the forgeries $(\sigma, \sigma^{'})$. It then uses these to extract the discrete logarithm of ω.

Our security proof utilizes the General Forking Lemma by Bellare and Neven [2] to reduce the security of our signature to the security of the Discrete Logarithm Problem (DLP) in $\mathbb{G}$. We end up proving that if F can forge the signature with probability *acc*, then DLOG problem can definitely be solved with a probability ϵ. However, since solving the DLOG is hard, this result implies that forging a signature in our scheme will be hard as well. We provide a quick understanding of our proof here:

Theorem 2. *If the discrete logarithm problem in G is $(\tau^{'}, \epsilon^{'})$-hard, then our signature scheme over G with n signing participants, a threshold of t, and a pre-processing batch size of π is $(\tau, n_h, n_s, \epsilon)$-secure whenever:*

$$\epsilon^{'} \leq (\epsilon^2)/((2\pi+1)n_h+1)$$
$$\textit{and } \tau^{'} \geq 4\tau + 2(n_s + 2\pi n_h + 1)t_{exp} + \mathcal{O}(\pi n_h + n_s + 1)$$

Proof. First we embed the challenge value ω into the group public key P . Our extractor algorithm $\mathcal{E}$ then uses the generalized forking algorithm to initialize our simulator $\mathcal{A}$ as $\mathcal{A}(P, h_1, ..., h_{n_r}; \beta)$, providing the group public key P, outputs for $n_r = (2\pi+1)n_h+1$ random oracle queries $h_1, ..., h_{n_r}$ and the random tape β. $\mathcal{A}$ then invokes the forger $\mathcal{F}$, simulating the responses to forger's random oracle queries by providing values selected from $h_1, ..., h_{n_r}$ and also acting as the honest party i_t in the *KeyGen*, *PreNonceGen* and *PartSign* phases.

$\mathcal{A}$ needs to trick forger by simulating signing of i_t without knowing its secret share of the key. For this,$\mathcal{A}$ generates a commitment and signature for participant i_t. To guess the challenge to return for a particular commitment when simulating a signing operation, $\mathcal{A}$ forks $\mathcal{F}$ to extract for each participant controlled by F, and consequently can directly compute its corresponding nonce. This is achieved by using the signers' commit of their pre-nonces. $\mathcal{A}$ who sees all random oracle queries, can therefore look up the commits before $\mathcal{F}$ can, and can thus correctly program the RO.
Once $\mathcal{A}$ has returned a valid forgery ($\sigma = (S, R)$) and the associated random oracle query index J, $GF_{\mathcal{A}}$ re-executes $\mathcal{A}$ with the same random tape and public key, but with fresh responses to random oracle queries $h_1, . \cdot h_{J-1}, h'_J., h'_{n_r}$ where $h'_J, ..., h'_{n_r}$ are different from previous inputs.
This effectively forks the execution of $\mathcal{A}$ from the J_{th} RO query. Given a forger $\mathcal{F}$ that with probability ϵ produces a valid forgery, the probability that $\mathcal{A}$ returns a valid forgery for our signature is ϵ, and the probability of $GF_{\mathcal{A}}$ returning two valid forgeries using the same commitments after forking $\mathcal{A}$ is roughly ϵ^2/n_r (ignoring the negligible portion).
The time taken to compute this comes out to be:
$4\tau + 2(n_s + 2\pi n_h + 1)t_{exp} + \mathcal{O}(\pi n_h + n_s + 1)$.
The running time for extractor E to compute the discrete logarithm by procuring two forgeries is 4τ (four times that for F because of the forking of $\mathcal{A}$, which itself forks F) plus the time taken by $\mathcal{A}$ for computing the signature and RO queries with additional operations $2(n_s + 2\pi n_h + 1)t_{exp} + \mathcal{O}(\pi n_h + n_s + 1)$.

5 Conclusion

Current blockchain transaction consist of the spending amount, hash of all transactions of previous block and the approving party's digital signature to allow spending. For increased security, Bitcoin-based chains allow for multi-signatures for spending. Any subgroup of participants can validate the transaction. In a practical network as dynamic as a blockchain network, where participant availability can not be guaranteed, an asynchronous, concurrent threshold signing protocol fulfils a very crucial need.
Our work improves upon these specific aspects and provides an alternative approach to existing protocols while being $EUF - CMA$ secure even with $t - 1$ corrupt signers as long as solving DLOG is hard.
We are presently working on reducing the communication required in **PreNonceGen** phase, while still maintaining secure asynchronous concurrent signing.

References

1. Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system. Manubot, 2019.
2. Bellare, Mihir, and Gregory Neven. "Multi-signatures in the plain public-key model and a general forking lemma." Proceedings of the 13th ACM SIGSAC conference on Computer and communications security (CCS). 2006.

3. Tomescu, Alin, et al. "Towards scalable threshold cryptosystems." 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.
4. Seurin, Yannick. "On the exact security of schnorr-type signatures in the random oracle model." Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Berlin, Heidelberg, 2012.
5. Shoup, Victor. "Practical threshold signatures." International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Berlin, Heidelberg, 2000.
6. Doerner, Jack, et al. "Secure two-party threshold ECDSA from ECDSA assumptions." 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.
7. Gennaro, Rosario, Steven Goldfeder, and Arvind Narayanan. "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security." International Conference on Applied Cryptography and Network Security (ACNS). Springer, Cham, 2016.
8. Gennaro, Rosario, and Steven Goldfeder. "Fast multiparty threshold ECDSA with fast trustless setup." Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). 2018.
9. Lindell, Yehuda. "Fast secure two-party ECDSA signing." Annual International Cryptology Conference (CRYPTO). Springer, Cham, 2017.
10. Lindell, Yehuda, and Ariel Nof. "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody." Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). 2018.
11. Bünz, Benedikt, et al. "Bulletproofs: Short proofs for confidential transactions and more." 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.
12. Maxwell, Gregory, et al. "Simple schnorr multi-signatures with applications to bitcoin." Designs, Codes and Cryptography 87.9 (2019): 2139-2164.
13. Boneh, Dan, et al. "Aggregate and verifiably encrypted signatures from bilinear maps." International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Berlin, Heidelberg, 2003.
14. Boneh, Dan, Xavier Boyen, and Hovav Shacham. "Short group signatures." Annual International Cryptology Conference (CRYPTO). Springer, Berlin, Heidelberg, 2004.
15. Komlo, Chelsea, and Ian Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold signatures." Selected Areas in Cryptography (SAC), 2020.
16. Nick, Jonas, Tim Ruffing, and Yannick Seurin. "MuSig2: Simple two-round schnorr multi-signatures." Annual International Cryptology Conference (CRYPTO). Springer, Cham, 2021.
17. Benhamouda, Fabrice, et al. "On the (in) security of ROS." Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Cham, 2021.
18. Pedersen, Torben Pryds. "A threshold cryptosystem without a trusted party." Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques (EUROCRYPT). 1991.
19. Kate, Aniket Pundlik. "Distributed Key Generation and Its Applications". Dissertation. University of Waterloo, 2010.
20. Boneh, Dan, Manu Drijvers, and Gregory Neven. "Compact multi-signatures for smaller blockchains." International Conference on the Theory and Application of Cryptology and Information Security (EUROCRYPT). Springer, Cham, 2018.

21. Chiesa, Alessandro, et al. "Decentralized anonymous micropayments." Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Cham, 2017.
22. Micali, Silvio, Kazuo Ohta, and Leonid Reyzin. "Accountable-subgroup multisignatures." Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS). 2001.
23. Shamir, Adi. "How to share a secret." Communications of the ACM 22.11 (1979): 612-613.
24. Schnorr, Claus-Peter. "Efficient identification and signatures for smart cards." Conference on the Theory and Application of Cryptology (CRYPTO). Springer, New York, NY, 1989.
25. Chor, Benny, et al. "Verifiable secret sharing and achieving simultaneity in the presence of faults." 26th Annual Symposium on Foundations of Computer Science (SFCS 1985). IEEE, 1985.
26. Feldman, Paul. "A practical scheme for non-interactive verifiable secret sharing." 28th Annual Symposium on Foundations of Computer Science (SFCS 1987). IEEE, 1987.
27. Canny, John, and Stephen Sorkin. "Practical large-scale distributed key generation." International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Springer, Berlin, Heidelberg, 2004.
28. Gennaro, Rosario, et al. "Secure applications of pedersen's distributed key generation protocol." Cryptographers' Track at the RSA Conference. Springer, Berlin, Heidelberg, 2003.
29. Drijvers, Manu, et al. "On the security of two-round multi-signatures." 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019.
30. Stinson, Douglas R., and Reto Strobl. "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates." Australasian Conference on Information Security and Privacy (ACISP). Springer, Berlin, Heidelberg, 2001.