Building Non-interactive Asynchronous Threshold Signatures For the Blockchain Ecosystem

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in ***Computer Science and Engineering***
by Research

by

Snehil Joshi
201407552
snehil.joshi@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
April 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Building Non-interactive Asynchronous Threshold Signatures For the Blockchain Ecosystem" by Snehil Joshi, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. Kannan Srinathan

To Mum

Acknowledgments

No man is an island and certainly my thesis would have not been possible without the support of all the people in my life. So as this chapter of my life comes to an end, I would like to appreciate and thank everyone who have helped me over these years in both my professional and personal life.

First of all, my advisor, Dr Srinathan for being not just a professional mentor but also a great human being to look up to. He has been a guiding light as a researcher and also as an incredibly intelligent, downright humble and compassionate person that I wish to emulate. In fact, I had planned to pursue ML research before I took his class in Advanced Algorithms and was so impressed with the subject and the teacher that in a week decided to switch over to his lab - probably the best decision of my life.

Next, I would thank the IIIT staff, who work tirelessly behind the scenes and manage the unbelievably complex system that is the IIIT research programme. They are the unsung heroes behind every successful graduation. Mr Manohar and Mr Karthik for managing CSTAR lab, Ms Pushpalatha for handling the thesis submission and defense process and Mr Y. Kishore for the dozens of mails and phone calls he answered to clarify my doubts. They have been not just been efficient professionals but also very understanding human beings who have always prioritised the student's welfare.

My colleagues from CSTAR: Durgesh and Shubham for endless hours of work put in the lab on research and paper writing and our bonding over blockchain, cryptocurrency and entrepreneurship. I hope our friendships will last long after our graduations. Jatin, Meher, Debarishi, Satvik and Tushant for the long discussions we had about careers and food and the Ultimate Question of Life, the Universe, and Everything. Charan and Manika for the Mario Kart sessions and Harsh for guiding me through the arduous thesis submission and defense process.

All my friends: Dharmateja, who I followed from BITS into MS in IIIT and whose advice kick-started this journey, Nauseen, the first friend I made in IIIT, Pramod, the troublemaker brother I never wanted and somehow got stuck with and have learnt to love, Shaheen, the one smart responsible person every group needs, Raghav, the supportive friend everyone wishes for, Sneha, the intelligent sister who somehow ended up marrying Pramod, Malu, the chill dude who introduced me to campus history and CCC, Akshita for teaching me to be more organized, Ronak, for showing me how to be resilient, Tejas, the only CogSci guy who spent more time than me at his lab, Aditi for being the only other pahadi kid on campus, Rahul for being the Man-U fan losing to Liverpool all the time, Abhinav, for reminding me how my life could be a lot worse if I worked in his lab, my batch-mates, Aditya, Vijay, Yashaswi(Bhaddu),

Somyajit, Sayantan, Bhargav, Ganesh and Karthik for being awesome bros. And finally CCC that helped me make so many two and four-legged friends in Hyderabad.

My family: Dad and my brother Swapnil, for being my pillars of strength and believing in me in every way a family possibly can. My Mom, Mamu and Nanu, who I wish were here to see this come to fruition, and who I am sure are really proud of me wherever they are. James, my best friend, for supporting me even when he didn't know how to. My in-laws for being the supportive and loving second family I never imagined I would have. And finally, my wife Shruti, for being an awesome friend and life partner who has kept me smiling through the toughest of times. Words will never be enough to thank her for her unconditional love and support.

Abstract

In the blockchain ecosystem, threshold signature schemes hold a special value. They 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. This provides not only added decentralization but additional security benefits as well.

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, in turn, results in a bottleneck due to during the various stages of the protocol. This thesis, aims to construct a threshold signature scheme that removes this dependence.

We achieve this by introducing non-interactive and truly threshold signatures. This implies that once the message to be signed is revealed, the individual signers can simply sign and broadcast their signature. These individual signatures can then be combined easily to construct the signature for the group without any further involvement.

Additionally, the signature scheme also uses misbehavior detection to impose accountability for invalid signing. This is done by adding an optional component to the individual signatures that can be removed while calculating the group signature but later invoked for detection.

Finally we prove that our scheme is safe against known distributed attacks and has *Existentially Unforgeabilility under Chosen Message Attack (EUF-CMA)* security in the Random Oracle Model for up to $t - 1$ malicious participants.

Contents

List of Tables

Chapter *1*

Introduction

1.1 Motivation

A few years back, I had the privilege to listen to a talk by Dr Martin Hellman. Naturally, I expected the talk to be about theoretical cryptography and heavily mathematical in nature. But to my, and everyone else's surprise, it was not. There was mathematics, of course, but Dr Hellman also brought to our attention, how political the nature of his work had been. Published in 1976, Dr Hellman had developed, along with Whitfield Diffie and Ralph Merkle, the very well-known, *Diffie-Hellman Key Exchange* (DHE) protocol [52]. This result became very famous, and rightly so. Before DHE, two parties that wanted to establish a secure encrypted connection between them were required to exchange some secret information (like their keys) using insecure means. However, using DHE, now that could be achieved without meeting as long as they could pass information to each other over a listen-only channel.

Needless to say, along with the academic world, the entire political and military world was also shocked. The US government realised that anyone with even decent computing power could now encrypt state secrets and send them outside the US right under the nose of the government agencies and they could not do anything about it. The Cold War was still ongoing and the role cryptography had played in WW2 was still fresh in everyone's mind. In fact, when the DHE algorithm came out, the export of cryptographic technology was still banned since cryptography was on the U.S. Munitions List as an Auxiliary Military Equipment.

Dr. Hellman then elaborated how the military's highest ranks approached him to stop publishing his research in the public domain. They feared, they told him, that the power of encryption in the hands of the many would lead to a disaster for the military intelligence community. Cryptographic power should always be centralised and government controlled, in their opinion. Over the period of the next few months, the Stanford legal team met military advisors and explained to them how universal encryption was good for all including the military and not just for state enemies. It was only after much deliberation that their work was eventually published. Incidentally, shortly after this, another equally famous result of the RSA [53] was also published, thereby introducing the use of cryptography in the public domain.

The reason I talk about this is not in order to explain any moral reasoning of why I ended up working in cryptography. I try to be as apolitical as one can get in my daily life. Instead, I want to point out the essential political nature of cryptography even for apolitical researchers. As a matter o fact, the eventual legal decision to include software source code as a form of free speech started with the landmark *Bernstein v. United States Dept. of Justice* case where Daniel J Bernstein argued that he be allowed to publish code for his encryption system *Snuffle* for public access. In fact, in the post-Snowden era, even as you read this, a tug of war in going on between the encryption-providers and encryption-breakers all over the world across the fields of mathematics, politics, law and media. What DHE was to the world of cryptography, in the year 2008, a similar result was published in the field of decentralised internet and banking.

Like the present state of the internet, the world banking system is centralised around a handful of powerful entities. This leads to a lot of situations where the authorities might make decisions on behalf of the citizens money, which are inherently not in the best interests of the citizen but in favour of other elites - a concept we call corny capitalism. An example would be the global financial crisis of 2007-2008 caused by overzealous bankers and their subsequent bailout of with public money without consultation from the citizenry. In a similar vein, western economic sanctions of certain countries force immense difficulties onto the citizens in obtaining necessary goods and services. And in war-trodden or economically declining nations the currency can lose value quickly people lose their life's savings overnight. The root causes for these issues are many, but an underlying aspect in all of this is the highly centralised control of the generation and transfer of financial assets. Unfortunately there was no viable alternative to this system until 2008.

That was the year, an internet pseudonym called "Satoshi Nakamoto" launched what was to become the world's first widely adapted decentralised currency: Bitcoin [1]. It provided with anyone with the means to generate, buy and sell an alternate currency that was not governed by any government or centralised authority. After almost a decade of being traded between tech-enthusiasts and hackers, it eventually went mainstream in 2017 and took the world by storm. Once the rest of the world realised the potential of this work and what they could build on it, it led many to believe and invest into their version of the future of the internet. And as as it goes with research and industry influencing each other, the world of blockchain significantly affected the way the world was undertaking research in the fields of distributed systems and cryptography - and in a more relevant aspect - my thesis research.

Before 2017, cryptography research was mainly the domain of academics or hardcore security professionals. Given how rigorous the research has to be and the time taken for adoption of any new protocol in security, the fruits of research used to ripen after decades. Most cryptographers didn't see their theoretical research being used in mainstream until after retirement. But once the crypto market boomed, it all changed overnight. The technology required to make blockchains work was heavily dependent on the two pillars of distributed systems and trust-less cryptography, and as the industry grew so did the adoption of these two techs. It opened up the whole wide world of distributed protocols that required "advanced" cryptography to be directly implemented into usable products. Suddenly from

using 30 year old production protocols, we now had multi-party computation, zero-knowledge proofs, homomorphic encryption and consensus protocols that were being adopted by the industry in a never before pace. This was the point where my work on these topics met the more practical work in the tech-industry.

1.2 Blockchain Technology and Distributed Signatures

A blockchain is essentially as the name suggest: a chain of blocks. Each block is nothing but a compilation of transactions along with the hash of the previous block. When a new block has to be added to the existing chain, the transactions plus some auxiliary information is compiled and hashed and sent for appending. Once it has been selected, it will be added on to the existing chain At any point in time, there might be several competing versions (called forks) of the same chain, with only one being the valid one. The methods for hashing, selecting which block is to be added next, selecting which is the valid chain to add to etc. might vary between the different chains and their technology, but the essence is the same: a block of transactions are compiled and appended onto the existing chain. To get a better understanding of my work, we also need to understand blockchain technology from a transaction point of view.

1.2.1 How a transaction proceeds on a blockchain network

The content of the block is transactions between various entities that are participating in that blockchain network. Let's say Alice wants to send Bob some cryptocurrency. Alice will first ask Bob for his account into which the transfer will be made. This will be Bob's public key (or a value derived from it). Then Alice accesses her wallet and using it transfers the amount from her address to Bob's. The transaction will have three main portions: the origin (Alice's signature derived from her private key), the recipient (Bob's address derived from his public key) and the amount to be transferred. There might be other auxiliary information like transaction fees and timestamp etc which we can ignore for now. Once this has been sent to the network, in the next step the network automatically verifies if the transaction is originating from the rightful owner. For this, it will crosscheck the digital signature included in the transaction against the sender's public key. If the transaction is validated, it enters the *mempool*, which is a buffer list of transactions awaiting approval on the network. The transaction waits here till it is selected after which it is compiled into a block that is added onto the blockchain. Once the block containing this transaction becomes a part of the main chain, the transaction is officially approved and the designated crypto amount is deducted from Alice and added to Bob.

1.2.2 Value of a distributed digital signature

As if obvious from the previous section, a transaction requires the digital signature of the sender to be approved. This is very similar to the current banking system where actual physical signatures or a

digital signature via account access is required to transfer money. However, in case of blockchains, the importance of this can not be stressed enough because unlike traditional banks which are open to legal arbitration in case of frauds, blockchains are decentralised immutable ledgers where transactions once approved are forever binding.

Therefore, if the digital signature is provided by a single entity, it introduces a single point of failure or compromise in the system. This is an antithesis to the idea of the a blockchain being a decentralised system to remove traditional banking with its single points of compromise. Therefore it is imperative to provide alternatives to access finances on the blockchain with similar resilience and security as the blockchain networks. This is where the domain of distributed digital signature comes in.

1.2.3 Distributed signing for blockchain

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. While most of them provide for the *n-out-of-n* (n, n) signing, which does remove a single point of failure as all the n entities need to be compromised in order to forge a signature it also opens them to the problem where a single entity being out of action prevents legitimate access to transactions. To prevent this, there are *t-out-of-n* (t, n) threshold signatures, which are protocols that allow a subset S of the players to combine their shares and reconstruct the private key to sign a message if $|S| >= t$, but disallow 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 required threshold value t.

While the research on threshold signatures has been around for decades, it has seen an increased interest since the advent of blockchain technology after Nakamoto's white-paper on Bitcoin [1]. However, even so, 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 generally any application requiring distributed permissions. This is what I have done.

1.3 Contributions

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.

Table 1.1: 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**

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 1.1 illustrates these properties.

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.

1.4 Organization of the Thesis

The following is a brief overview of the upcoming chapters of the thesis.

Chapter 1 was the introduction for the thesis. It provided with the motivation for the work as well as the path that led us to it. **Chapter 2** builds the basics required for getting into the research work done in the thesis. Since our work is very multi-disciplinary, this will include broadly, the relevant topics in distributed systems as well as cryptography and their connection to our current work. . It also serves as a literature survey and history of the work done before us. In **Chapter 3** we formally define our problem statement and describe the various definitions and assumptions we have made and the reasoning behind them. This will make it easier to follow our work and results in the next chapter. **Chapter 4** presents our work and results as the protocol, "*Asynchronous Truly-Threshold Schnorr Signing for Inconsistent Availability*" - hereby also labeled as the acronym *ATSSIA*. **Chapter 5**, gives the proofs for correctness and security for our protocol. **Chapter 6** is the conclusion with a brief cost analysis, summary of our results, comparison with existing protocols and present scope for future research in the a similar direction.

Chapter *2*

Background and Related Work

To understand how threshold signatures work, we first need a brief understanding of distributed systems and consensus protocols. Distributed systems involve a network of nodes - alternatively called agents, entities, individuals, participants, or processes - coming together to accomplish a single task such that it appears to have been done by a single entity. In order for such a system to work, it is usually required for the various nodes to agree on the same value for some given variables at various points in the execution. Having a sound mechanism to achieve an agreement on a value becomes even more important in case we expect some nodes to fail during the task. Some of these failures can be fail-stop or semi-honest or malicious. Mechanisms designed to handle such situations are called consensus protocols.

2.1 Consensus Protocols

Consensus protocols have been a hot topic of in distributed systems for almost half a century. To start with, the seminal work of Lamport et al. [31] and Fischer et al. [32] effectively defined the consensus problem in distributed systems.

The paper by Lamport et al. [31] is the famous Byzantine Generals Problem wherein the problem of consensus was represented as generals (the nodes) of a distributed army (the distributed network) containing possible traitors (the malicious nodes) trying to achieve agreement on whether they should attack the enemy or not. It showed that, using only oral messages, consensus could be achieved if and only if less than a thirds of the generals were traitors. This could be drastically improved upon using unforgeable written messages, where the problem is solvable for any fraction of traitors. The result from [31] however, assumed a synchronous setup. Fischer et al.'s result in [32] proved that it is impossible for any deterministic protocol to achieve consensus in an asynchronous model in the presence of even a single crash. While this looked like a major setback to the field of asynchronous consensus, it opened up numerous creative ways of researchers achieving consensus using other techniques like randomization

as seen in [33, 34, 35, 36]. For an excellent short study of the entire topic, we would strongly recommend reading Fischer's survey of Byzantine fault tolerance, which though old, is still highly relevant [37].

All the above results worked in a strictly mathematics based network approaches to solving the problem. In 2008, however, the Bitcoin consensus protocol [1] came up with an alternative mechanism called proof-of-work to achieve an agreement on which block is to be added next onto the chain. Proof-of-work is a proof based wherein the prover has to convince the verifier that a certain amount of computation has been done in order to solve a reasonably hard problem [38]. In case of Bitcoin, the prover has to provide a hash of the block it wants to add in a certain format. The node that is able to first generate its block's hash in this particular format gets to add the corresponding block onto the chain.

Since generating the hash value takes a lot of computation power, proof-of-work results in a lot of wasted energy. To prevent this, alternatives to proof-of-work have been explored in recent times. Some of the major ones are Proof-of-Stake, Proof-of-Time, Proof-of-History.

Just as cryptography provides new approaches to solve problems in distributed systems, so do distributed algorithm provide for solutions in cryptography protocols. One such sub-field of cryptography that uses disturbed algorithms is the domain of secret sharing.

2.2 Threshold Schemes

2.2.1 Secret Sharing

Secret sharing is the domain of cryptography dealing with distributing shares for a secret between participants such that individually each share doesn't reveal any information about the secret but when combined sufficiently, the secret can be constructed. The simplest secret sharing scheme can be thought of two parties P_1 and P_2 having shares for secret S as s_1 and s_2 respectively, such that $s_1 \oplus s_2 = S$. As is apparent, without knowing each others shares, it is information-theoretically impossible to obtain the secret S. This can be trivially extended to the (n, n) case where n such parties each having a share combine to give $s_1 \oplus s_2 \oplus s_3 ... \oplus s_n = S$. Again, as long as even one share is not provided, the knowledge of all other shares doesn't help reveal the secret.

While the (n, n) solution is good in itself, it can have rather limited applications. Therefore, we need to modify it considerably in order to incorporate other desirable options like threshold and proactive properties.

2.2.2 Threshold (t, n) Secret Sharing

Threshold (t, n) secret sharing protocol is one that enables distributing n shares such that t-1 shares are not enough to construct back the secret but with t shares it is possible. WLOG, in case of the Shamir sharing scheme, it is formally defined as: 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].

The concept of threshold secret sharing schemes was developed independently in two separate approaches by [23] and Blakley [40]. Shamir's scheme uses polynomial interpolation while Blakely's uses intersecting planes. Along with these, the Chinese Remainder Theorem has been used to create the Asmuth-Bloom's [41] and Mignonette's [42] secret sharing schemes. Shamir's secret sharing protocol being the most efficient of the lot is the one most in use. Intuitively, the all the approaches follow a similar pattern, with the main difference being the way the shares are generated.

2.2.3 Verifiable Threshold (t, n) Secret Sharing

A verifiable secret sharing requires additional information from the dealer during the share distribution in order to prevent a malicious dealer from giving incorrect shares to the players [24]. Like a regular secret sharing scheme, a (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. In addition to this, the dealer also commits information wrt the secret shares that can enable any player to verify the consistency of their share. These commitments should not leak any knowledge of the secret or the share. After receiving the share and commitment values, a player can cross-check the share value and either approve or reject the sharing phase. If approved by all the honest participants, the sharing phase is deemed successful. 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.

VSS schemes were first introduced by Benny et al [25]. Some commonly used schemes are by Feldman [26] and Benolah [39]. The Pedersen [18] VSS scheme has largely replaced Feldman's since it has improved security by removing information leakage via commitment values.

As we will see later, VSS is essential for our distributed scheme where the secret key generation phase can not have a single trusted party and therefore each individual node has to effectively act as a distrusted dealer and distributively generate shares for everyone.

2.3 Digital Signatures

A digital signature is a cryptographic mechanism to validate and authenticate the origin of digital documents [43]. More specifically, a digital signature employs asymmetric cryptographic schemes to allow the signer to combine a given message *m* with their private key *k* in a cryptographically secure

way such that anyone with the signature can and the signer's public key *P* can verify the authenticity of the signature.

Definition 2.3.1. Formally, a digital signature is a scheme consisting of three *Probabilistic Polynomial Time* (PPT) algorithms:

- *Key Generation:* A key generation algorithm that takes certain public parameters (pp) and generates a private-public key pair (x, P) from it. The private key is secret privy only to the signer while the public key can be made available to anyone who signature wants to verify the signature.

- *Signing:* The singing algorithm that returns a signature σ from given inputs: messages m and private key x.

- *Verification:* The verification algorithm takes the output from the signing algorithm and verifies it using the value of output σ, message m and public value P. It either accepts or rejects the validity of signature.

2.3.1 Properties of Digital Signatures

For a digital signature scheme to work it should satisfy a minimum of these two properties:

- *Correctness:* The property of correctness is satisfied when if we sign the message using the private key generated using the KeyGen algorithm, then verification algorithm will *always* result in an accept. More formally:

$$\Pr\left[(P, x) \leftarrow \text{KeyGen}(pp), \text{Verify}(P, m, \text{Sign}(x, m)) = \text{accept}\right] = 1$$

- *Security:* A digital signature is considered $S(\sigma, \tau, Q)$ secure against forgery by a PPT adversary $\mathcal{A}$ running for time at most τ and making at most Q queries to the signing oracle S, if the probability of generating a previously unseen and valid signature is:

  $$\Pr\left[(P, x) \leftarrow \text{KeyGen}(pp), (x, \sigma) \leftarrow \mathcal{A}^S(P, pp), x \notin Q, \text{Verify}(P, x, \sigma) = \text{accept}\right] < \epsilon$$

  This specific criteria of security against forgery is called **EUF-CMA**. It will be explained in detail in the later chapters.

While threshold schemes exist for most types of standard signature schemes, we use the Schnorr signature scheme as our base since due its linear design has made it the current standard for research work in signature schemes in blockchains.

2.3.2 The Schnorr Signature Scheme

The Schnorr signature scheme consists of three algorithms: *KeyGen*, *Sign* and *Verify*. For a message m the Schnorr signature [24] is calculated as follows:

Protocol: Schnorr Signature Scheme

KeyGen:

1. Select $\mathbb{G}$, an elliptic curve group of prime order q with generator G. Also select H be a cryptographic hash function mapping to Z_q^*.
2. Choose random value $x \in Z_q$.
3. Calculate the private key $P = x \cdot G$, where the $(\cdot)$ operator denotes scalar multiplication on the elliptic curve.
4. The private-public key-pair is (x, P).

Sign:

1. Choose random nonce $k \in Z_q$
2. Calculate public commitment $r = k \cdot G$
3. Calculate $e = H(P, r, m)$
4. Calculate $s = k + e \cdot x$
5. The final signature is $\sigma = (r, s)$

Verify:

1. Given values m, P, σ
2. Extract r and s from σ and compute $e_v = H(P, r, m)$
3. Calculate $r_v = s \cdot G - e_v \cdot P$
4. If $r_v = r$ then verifier accepts the signature as valid.

2.4 Threshold Signatures Schemes

In aggregate signature schemes, threshold signature schemes make use of the (t, n) security attribute of threshold protocols. This enables signers to generate a group signature over a message m using their shared secret keys, giving the final signature the appearance of having been generated using just one key and possessing all the characteristics of a typical digital signature. This enables signing without letting any one person in on the secret, and it also makes it so that the threshold aggregate signature may be verified just like a regular signature. In threshold signature systems, a single key P is used to represent the group's public key while a secret key x is dispersed among the n participants.

2.4.1 Centralised KeyGen

Some threshold signature schemes utilise a centralised key generation algorithm. Like threshold secret sharing schemes, this involves a trusted dealer - which can itself be single or a distributed entity - that will generate the public and private key shares for all the participants. The private key x, here serves the same purpose as the secret in the previously mentioned secret sharing schemes. Depending on the protocol design, the participants can then either combine their keys to first generate the private key and use it to sign the message or they can sign the message individually and then combine their individual signatures to generate a final signature for the entire set of participants.

If the dealer is trusted by the parties, a simple Shamir Secret Sharing will suffice. In case of an distrusted dealer, the parties can use a VSS scheme.

2.4.2 Distributed KeyGen

Unlike the centralised dealer approach, which opens the protocol to a single point of failure or compromise, most threshold schemes prefer to generate their shares distributively [26]. In this approach, the participating nodes exchange messages with each other to individually generate their private share as well as the secret key for the entire group. Algorithms that enable this are called Distributed Key Generation algorithms or DKGs for short. The generic format of a DKG has two phases similar to VSS schemes. In the sharing phase, each node acts as a dealer for all other nodes and distributes its own random value as shares among them. At the end of this, each party will have n shares values, with each share corresponding to a secret of each of the n participating nodes. In the reconstruction phase, the nodes combine their share values they using some predefined reconstruction function to generate the public key P for the entire set of nodes. The private key x is not reconstructed at any point in this protocol making it more secure.

For security reasons that we will show later, some DKGs require more than just two phases. Several threshold signature schemes currently in use are MuSig [12] and FROST [15]. However, not every signature scheme can be transformed into an equivalent threshold version easily. For example, ECDSA and is one of the most notorious schemes to convert [6, 7, 8, 9]. At the same time, we have BLS

[13], which can be directly used in a threshold scheme out of the box. But given that most existing blockchain platforms have good support for Schnorr and ECDSA signature schemes and Schnorr is much more simple to modify than ECDSA in this context, for practical purposes, we chose the Schnorr signature scheme as the basis for our protocol.

2.5 Attacks on Distributed Signatures

We briefly describe the following attacks on multi-party protocols in a distributed setting. These attacks will be detailed in a later chapter during the reasoning for the various portions of the protocol.

2.5.1 Rouge Key Attack

The rogue key attack [54] is possible in case of multi-party protocols where the adversary, in control of one or more participants, is allowed to view the inputs supplied by the honest parties before submitting its own inputs. This allows it to modify its own inputs to adapt to the available information from other inputs in order to subvert the protocol during the key generating phase. We counteract this by committing the inputs before using them in the protocol.

2.5.2 Drijvers' Attack

The Drijvers' attack [29] is a modification of the Wagner's sub-exponential k-list attack [44] which can be used to forge a multi-party signature. It relies on replacing the sum of T individual hash outputs in a parallel execution of a multi-party function with a single hash output. This attack is possible, if the adversary has the capability to choose the message m to be signed or to adaptively select its own individual commitments used to calculate the group commitment value after seeing commitments from all other signing parties. We therefore use an adaptive commitment function that binds the message m to the commitment value such that if one changes the other changes as well.

2.6 Additional Cryptographic Primitives

Apart from the aforementioned definitions, our work utilizes several other cryptographic primitives as parts of our protocol. While it is not necessary to know their detailed inner workings, it is still beneficial to have an idea about them in order to better understand how and why they fit into our scheme.

2.6.1 Commitment Schemes

Commitment schemes allow a sender to bind some value without revealing it. Later at a later stage the sender can reveal the value to the receiver and the receiver can verify that the sender did not cheat by

sending something else. A bit commitment essentially allows one to commit a chosen value from a list of options while keeping it hidden from others, but with the ability to reveal the value later. Interactions between two parties happen in two phases: 1. Commit: the value is chosen and committed such that the sender is now bound to this choice but the value is hidden from the receiver 2. Reveal: the committed value is revealed to the receiver and checked to be correct The concept of bit commitment was first mentioned by Manuel Blum [45]. It was used in several important works [46, 47] before it was formalized by Brassard, Chaum and Crepeau [48].

2.6.2 Zero-Knowledge Proofs

A Zero-Knowledge Proof (ZKP) is a way for a prover to convince a verifier that an assertion being made is correct with a very high probability without actually revealing anything else about the assertion. A zero-knowledge proof must satisfy three properties [50]:

- *Completeness*: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
- *Soundness*: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
- *Zero-knowledge*: if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

The wide applicability of zero-knowledge proofs, was first demonstrated by Goldreich, Micali and Wigderson [49], who showed how to construct zero-knowledge proof systems for any NP-set, assuming one-way functions and using any commitment scheme based on them. This was a turning point in cryptography as it demonstrated the generality as well as the wide applicability of zero-knowledge proofs in cryptographic protocols. Goldreich and Oren [51] introduced auxiliary-input ZKP, where the verifier has an additional information called auxiliary information prior to the commencement of the protocol. This is a more general model of the ZKP which also proves the existence of sequential zero-knowledge proofs.

2.7 Generalised Forking Lemma

In order to prove the security of our threshold signature scheme, we utilise the *Generalised Forking Lemma* [2] since it is used for proving security for the underlying Schnorr signature scheme. In common terms, the lemma states that in the random oracle model, after several queries to a random oracle if the adversary can forge a signature with non-negligible probability, then there is a non-negligible probability that the same adversary can create a second forgery in an attack with a different random oracle.

We design our algorithm such that two appropriately chosen forgeries result in solving the underlying hard problem. This means that if an adversary can break the overlying protocol (signature scheme in our case) with a non-negligible probability of success, then the underlying hard problem that it reduces - *elliptic curve discrete log problem (ECDLP)* in our case - can also be solved with a non-negligible probability. Conversely, this proves that *only if* the ECDLP can be solved with a negligibly low probability, the protocol based around it will also be unbroken with a similar probability.

The *Generalised Forking Lemma* uses the *Generalised Forking Algorithm* (GFA) to derive the security proof from the protocol. The generalised forking algorithm GFA associated with a randomised algorithm A is the algorithm that given input x proceeds as follows:

Algorithm Generalised Forking Algorithm

---

1. Fix integer $q \geq 1$ and a set H of size $h \geq 2$ and have A as a randomized algorithm that on input $(x, h_1, ..., h_q)$ returns a pair, (I, σ) where $I \in Z_q$ and σ is side output
2. Pick ρ random coins for A
3. $h_1, ...h_q \leftarrow H$
4. $(I, \sigma) \leftarrow A(x; h_1....h_q; \rho)$
5. If $I = 0$ then return $(0, \epsilon, \epsilon)$
6. $h'_I, ...h_q \leftarrow H$
7. $(I', \sigma') \leftarrow A(x; h_1, ...h_{I-1}, h'_I, ...h'_q; \rho)$
8. If $(I = I' and\ h_I \neq h'_I)$ then return $(1, h_I, h_{I'}, \sigma, \sigma')$ else return $(0, \epsilon, \epsilon)$

Lemma 2.7.1 (*Generalised Forking Lemma*)**.** If the accepting probability of A, denoted by acc, is defined as the probability that $I \leq 1$ in the experiment: $x \leftarrow IG; h_1, ..., h_q \leftarrow H; (I, \sigma) \leftarrow A(x, h_1, ..., h_q)$ the Generalised Forking Lemma states that if the probability of a fork is, $fork = Pr[b = 1; x \leftarrow IG; (b, \sigma, \sigma') \leftarrow GFA(x)]$, then:

$$fork \geq acc \cdot (\tfrac{acc}{q} - \tfrac{1}{h})$$

Or alternatively:

$$acc \leq \tfrac{q}{h} + \sqrt{q \cdot fork}$$

Given the two forgeries (σ, σ') and oracle queries corresponding to these forgeries i.e., (h, h') we can extract the private key x of the signer. Our protocol is designed such that this private key will be the challenge to the ECDLP and will therefore result in the adversary being able to solve the hard problem of ECDLP but only if we assume that it can forge the signatures.

Chapter *3*

System Model

Before we explain our protocol we need to define the environment and conditions in which the protocol is expected to work. This also requires providing more details and formalising the problem statement. The modelling and assumptions given in this chapter are essential to understand the subsequent work.

3.1 Network Model

3.1.1 Network and Communication Model

Our protocol works largely in a dynamic distributed network such that while the set of eligible nodes is fixed at the beginning, after that the nodes are free to leave or join the network during the execution of the protocol without affecting the outcome. We assume that our network can support a reliable broadcast channel as well as secure *peer-to-peer* channels between the participating nodes. In a truly distributed network this means that the underlying graph is a complete graph.
Regarding the requirements for availability of nodes in the network, the initial phases for key and pre-nonce generation both require *all* participants to be present. However, the signing phase doesn't require the minimum quorum of $\geq t$ signing parties to be present at the same time.

3.1.2 Message Indexing

Since our protocol uses batch processing for generating the 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.

3.2 Adversary Model

First we need to carefully fix the power and scope of the potential adversary. This includes the computational power in terms of a single entity as well as distributed power in terms of fraction infected as well as honest or malicious behaviour.

3.2.1 Computational Power

Cryptographic applications generally involve two kinds of adversaries: computationally unbounded or computationally bounded. We need to provide information-theoretic security for protection against computationally unbounded adversaries. While this is a good theoretic requirement, in practice it is rather excessive for most cases and almost always results in a performance degradation due to overhead. Instead, a well-designed computationally secure scheme is enough for most practical cases. In addition to reducing unnecessary overhead, modelling a problem after bounding the computational power of the adversary enables us to focus on the additional approaches of the adversary that can be used to perform a more sophisticated attack instead of direct brute force. For example, we can check how our security definition of the protocol changes if an adversary manages to access ciphertext files for some known plaintext files (*Chosen Plaintext Attack or CPA*).

Therefore, for our protocol, we assume a computationally-bounded *Probabilistic Polynomial Time* (PPT) adversary. In addition to this, we assume the following additional powers of the adversary.

3.2.2 Corruption Power

Unlike a standalone network, a distributed network opens up a much larger attack surface for the adversary. In addition to existing attacks, in a distributed network, the adversary also has the power to corrupt a set of participating nodes. This corruption is possible either as a static or dynamic adversary. In case of a static adversary the adversary is assumed to control a fixed set of nodes at the beginning of the protocol which remains the same throughout. A dynamic corruption, on the other hand, can control variable sets at any point in the protocol, i.e., the set of compromised nodes might change during the proceeding of the protocol, as long as the union of those sets does not exceed the threshold value.

3.2.3 Honest/Malicious Type

Any multi-party protocol requires us to specify the behaviour of the potential adversary: semi-honest or malicious. A semi-honest adversary is one that will follow the rules set in the protocol. A malicious adversary on the other hand, can deviate from the protocol, e.g., changing the inputs in any phase, aborting the protocol, abruptly etc. As is evident, semi-honest adversaries are easier to handle than malicious ones. Generally we require additional steps in the protocol to ensure rule-abiding behaviour from malicious adversaries, i.e., we try to force them to behave like semi-honest ones. Our protocol is secure for an arbitrary number of malicious adversaries in the distributed setting.

3.3 Cryptographic Assumptions

3.3.1 Discrete Logarithm Problem

In cryptographic protocols, security of a protocol is usually proven by reducing that protocol to well known one-way function. This implies that breaking that protocol can be reduced to adversary being able to break a well known one-way function. The discrete logarithm problem is one such standard problem.

Let p be a prime whose bit-length is linear in input parameter k. Given a generator g of a multiplicative group G of order p and $x \in Z_p^*$, the discrete logarithm or DLog assumption states that $Pr[A_{DLog}(g, g^x) = x] \leq negl(k)$ for every polynomial-time adversary A_{DLog} and some negligible function $negl()$.

3.3.2 Elliptic Curves Discrete Logarithm Problem

For our work, we have used elliptic curves in place for standard discrete log. The hardness for the ECDLP is derived from DLog : the assumption being that it is infeasible to finding the DLog of random elliptic curve element from its base point. This is also called ECDLP for the Elliptic Curve Discrete Logarithm Problem.

Formally, given a prime p and $q = p^r$, an elliptic curve E over a field F_q is a curve of points (x, y) satisfying the equation $y^2 = x^3 + ax + b$ (where $a, b \in F_q$) along with O "the point at infinity".
Given a generator point G on E and point $R = k.G$ where k is a scalar and R belongs to the cyclic subgroup generated by P, the ECDLP assumption states that E is $(\tau, \epsilon) - hard$ on E if for any algorithm $\mathcal{A}$ running in time at most τ the probability of solving ECDLP problem is at most ϵ.

3.4 Security Model

In a way similar to encryption schemes, security for digital signature schemes is also modelled as a game-based approach. In this, the game replicates a scenario where a challenger who is an entity using

the digital signature scheme challenges the the adversary to break the scheme in a specific manner under a specific attack.

3.4.1 Forgery Types

There are four types of signature forgeries:

- *Total break (TB):* Total break is the type of signature forgery wherein the attack enables the adversary to recover the private signing key itself.
- *Universal Forgery (UUF):* In case of UUF, the adversary can forge signatures for every possible message it wants without necessary recovering the private signing key. In practice this implies that the challenger can chose a specific message m which the adversary will have to forge signature for.
- *Selective Forgery (SUF)*: In SUF, the adversary can forge a pre-determined message m that had been fixed for forgery before the attack. This has more freedom for adversary than in UUF, since this allows it to chose a message with helpful properties for the forgery.
- *Existential Forgery (EUF):* In case of EUF, the adversary can forge at-least one message m. The content of the message m here don't matter and can be even meaningless. This is the strongest model for forgery since it gives maximum freedom to the forger adversary.

As is evident from the above definitions, arranged in terms of freedom given to the adversary : $EUF \geq SUF \geq UUF \geq TB$. Conversely, this implies that if a protocol is secure for a preceding forgery, where the adversary has more freedom, it will be secure in the subsequent ones as well, where the extra rules restrict the adversary's behaviour.

3.4.2 Attack Types

Digital signatures can be required to provide protection against one of the following three attacks.

- *Key-Only Attack (KOA):* KOA is the most restrictive type of attack in which the only knowledge the adversary has is that of the public-key. This is the weakest attack since it implies that the adversary has no means of obtaining any extra information about the signature scheme etc.
- *Known-Message Attack (KMA):* In the KMA model, the adversary has, in addition to the public information like public key, access to a set of messages and corresponding signatures on them. The messages and signatures are assumed to be randomly provided.
- *Chosen-Message Attack (CMA):* The CMA attack model, builds upon the KMA model by allowing the adversary to chose the plaintext messages it wants to get signatures for.

3.4.3 EUF-CMA Security

The aforementioned forgery and attack types leave us with several options from the most restrictive TB-KPA (where the adversary has to recover the private key just from knowing the public key) to EU-CMA - (where the adversary can show forgery on any arbitrarily chosen message while having access to a set of valid signatures on its own choice of messages). We chose **EUF-CMA** as the security model for our scheme since it is standard for digital signature schemes as it allows the freedom for the potential adversary to behave very close to a real world one.

Definition 3.4.1 (EUF-CMA)**.** A signature scheme is considered **EUF-CMA** secure *iff* for any polynomial-time adversary A, the probability of winning the **EUF-CMA-GAME** is negligible.

Here the **EUF-CMA-GAME** is a defined as a game between the adversary and the challenger such that for a given signature scheme $(KeyGen, Sign, Verify, n)$, and adversary A with its private set of messages notated as M:

1. Let first A make multiple queries to $Sign(m)$ to obtain its required set of signatures for the messages in M. Initially $M = \emptyset$. At the end of this step, A will have some messages in M along with the corresponding valid signatures for them.

2. Knowing the public key of the singer as well as the signatures for the messages in M, A will output a signature σ for some message m such that $m \notin M$ and $Verify(\sigma, m) = accept$.

Chapter *4*

ATSSIA: Asynchronous Truly-Threshold Schnorr Signing for Inconsistent Availability

This chapter deals with the main protocol including all the pre-processing phases required for it. We start with the protocol overview followed by a detailed description of the various phases along with the reasoning for the choices made in each phase.

4.1 Overview

A single signer signature scheme has three main phases: *KeyGen*, *Sign* and *Verify*. In our protocol, since it needs the threshold property, we need to modify it by adding a pre-processing phase for nonce-generation *PreNonceGen* and also replace the *Sign* phase with two separate phases of *PartSign* and *Aggregate*.

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. We also provide the optional phase for misbehaviour detection in case of faulty partial signatures being generated.

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.

The following is a quick overview of our signature scheme:

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:
  $s_{ij} = ({k_{ij}}^a + h_j \cdot {k_{ij}}^b) + H_m(m_j, R_j, P) \cdot x_i + \rho \cdot H_m(m_j, r_{ij}, p_i) \cdot x_i$
  (Section 4.3 and 4.7 explain 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:
  $S_j \cdot G = ?R_j + H_m(m_j, R_j, P) \cdot P + \rho \cdot H_m(m_j, r_{ij}, p_i)) \cdot p_i$

4.2 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.Remember that Pedersen's DKG is essentially a parallel execution of Feldman's VSS by each participant acting as the dealer, from which they derive their secret share as the total of the shares they receive 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

Round 1:

1. Let there be n participants indexed as i where $i \in \{1, 2...n\}$.
2. Each i generates t random values $(a_{i_0}, \ldots a_{i_{t-1}}) \in \mathbb{Z}_q$ to be used 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.
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 $C_i =< A_{i_0}, \ldots, A_{i_{t-1}} >$, 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.

Round 2:

1. Each i broadcasts the secret share $(i, f_i(j))$, to every other participant P_j while keeping $(i, f_i(i))$ for itself.
2. Now each i verifies its shares by checking if $f_j(i) \cdot G = \Sigma(A_{j_k} \cdot (i^k \, mod \, q)), k = 0, 1 \ldots t-1$, with $\perp$ on failure.
3. Each i finally calculates its individual private key share by computing $x_i = \Sigma_j f_j(i), j = 1, 2 \ldots n$, and stores $x_(i)$ securely.
4. Additionally, each i also calculates its public share for verification $p_i = x_(i) \cdot G$, and the group's public key $P = \Sigma_j^n A_{j_0}, j = 1, 2 \ldots n$. The participants can then compute the public share for verification for all the other participants by calculating $p_j = \Sigma_j^n \Sigma_k^{t-1}(A_{j_k} \cdot (i^k \, mod \, q)), j = 1, 2 \ldots n, k = 0...t-1)$. 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: $(x_1, x_2 \ldots x_n)$
- Partial public key shares: $(p_1, p_2 \ldots p_n)$
- Group public key: P where $P = X \cdot G$

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.

4.3 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] constructed a threshold Schnorr signature protocol that utilized DKG to generate several nonce values in its pre-processing stage. This was independent of the 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_{ij}^a, r_{ij}^b). Each prospective signer then commits r_{ij}^a and r_{ij}^b 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:

$$r_{ij} = r_{ij}^a + h_j . r_{ij}^b$$

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, to sign up to π prospective messages, the **PreNonceGen** phase will generate $2 \cdot \pi$ pre-nonces (2 for each π messages) per participant using a total of $2 \cdot \pi$ parallel DKGs.

PreNonceGen

Round 1

1. Each participant i for each message j, computes k_{ij}^a value (similarly for k_{ij}^b) by generating t random values $(a_{i_0}, a_{i_1}..., a_{i_{t-1}})$ in Z_q.
2. Each i uses the a_{i_l} 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
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.

Round 2

1. Participant i receives (C_j, σ_j), $j \neq i$ from every other participant $j \neq i$ and verifies $\sigma_j = (w_j, c_j)$ by computing $h_j = H(j, S, A_{j_0}, w_j \cdot G)$ and checking for $w_j = c_j \cdot G - A_{j_0} \cdot h_j$ with $\perp$ (abort) on failure.
2. 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.
3. 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.
4. 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.
5. Each i also calculates the corresponding public share for nonce verification $r_{ij}^a = k_{ij}^a \cdot G$
6. The participants 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.

4.4 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 an individual signer with its participant-index i and other signers in the protocol with index o in order to avoid any confusion with the message index denoted by j. The values with each participant i are:

- 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
- π threshold pre-nonce commitment pairs (R_j^a, R_j^b) calculated from the values of (r_{oj}^a, r_{oj}^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 4.7*. We emphasize that this is an optional part of the signature scheme that is not a necessity and can be excluded depending on the requirements. It doesn't affect the core functionality or security of the rest of the protocol in a negative manner.

The **PartSign** phase is as follows:

PartSign

Given the j_{th} message $m_j \in Z_q$, hash functions $H_r(.)$ and $H_m(.)$ mapping to $\{0,1\}^l$ in Z_q^*:

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. 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 $\sigma_{ij} = (s_{ij}, R_j, r_{ij})$

4.5 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 4.7*.
   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.

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 4.6*:
   $(S_j \cdot G)\, mod\, \rho =? R_j + H_m(m_j, R_j, P) \cdot P$
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 4.7* 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-aggregated if at least t valid signatures are available:
   $S_j = \Sigma\lambda_{ij} \cdot s_{ij}$
6. The aggregator broadcasts the threshold signature as:
   $\sigma_j = (S_j, R_j)$

4.6 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 *(see Section 4.7.2)* before checking it. This is done in order to remove the individual signer's misbehaviour detection portion from the signature without altering the contribution of individual signatures.

The new verification phase is as follows:

Verify

Given values m_j, P, σ_j and ρ *(explained in 4.7.2)*, 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

4.7 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.

4.7.1 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:
$\sigma_{ij} = (s_{ij}, R_j, r_{ij})$

4.7.2 Choosing the ρ value

The ρ value should have these properties:

- It should be sufficiently large so that modulus of the signature with ρ still preserves the rest of the signature as before, i.e., $\rho > (k_{ij} + H_m(.) \cdot x_i) \forall i, j$
- 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.

4.8 Verification of the Threshold Signature

The inclusion of a misbehaviour detection part in the scheme does not change the way we verify the threshold signature. This is because after aggregating the individual signatures, when the verifier needs to verify the final signature, they can simply remove the individual identifier by taking the partial-signature modulus rho, i.e., $S = S_j mod \rho$ and thereafter follow regular verification on the resulting expression (see *Section 4.6* for details).

4.8.1 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 follows:

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

Chapter *5*

Proofs

For any signature scheme to work, should have two main properties: correctness and security. This chapter provides proofs for both these properties for our signature scheme ATSSIA. We prove our scheme's security of our scheme in the **EUF-CMA** security model against a *PPT* adversary. Theorem

5.1 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 5.1.1. 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 $H_r, givenapublic - privatekeypairandH_m$, 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. □

As shown above, since our signature scheme is essentially replicating a single-party Schnorr proof by substituting the threshold key and nonce values with single value ones, our scheme provides correctness at the same level as that of a single signer Schnorr protocol.

5.2 Proof of EUF-CMA Security

By demonstrating that the difficulty of forging our threshold signature by executing an adaptively chosen message attack in the Random Oracle (RO) model can be reduced to the difficulty of computing the discrete logarithm of an arbitrary challenge value ω in the same group, as long as the adversary controls only up to $t-1$ participants, we will prove security against existential unforgeability in chosen message attacks **EUF-CMA**.

Our proof uses these four 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 5.2.1. 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:

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

Proof. First we embed the challenge value ω as part of public key P for the group. Our extractor algorithm $\mathcal{E}$ then uses the $GF_{\mathcal{A}}$ to execute our simulator $\mathcal{A}$ as $\mathcal{A}(P, h_1, ..., h_{n_r}; \beta)$, providing the public key P for the group, outputs for $n_r = (2\pi + 1)n_h + 1$ random oracle queries $h_1, ..., h_{n_r}$ and supplying the random tape β. $\mathcal{A}$ then initiates the forger $\mathcal{F}$, simulating the responses to forger's random oracle queries through giving it values from the set $h_1, ..., h_{n_r}$ 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 it has to simulate the signing, $\mathcal{A}$ forks $\mathcal{F}$ to extract the secret for each participant under F, and thereby can easily compute the corresponding nonce value. 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 the simulator $\mathcal{A}$ has replied with a valid forgery $(\sigma = (S, R))$ and the corresponding random oracle query index J, $GF_{\mathcal{A}}$ re-executes $\mathcal{A}$ keeping random tape and public key P same, but with fresh response 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. Assuming we have a forger $\mathcal{F}$ that produces a valid forgery with probability ϵ, the probability that the simulator $\mathcal{A}$ returns a valid forgery for our signature is also ϵ, and the probability of $GF_{\mathcal{A}}$ returning two distinct 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 total running time for our extractor E in computing the discrete logarithm after extracting the forgeries is 4τ. This value is four times that for F because the extractor forks $\mathcal{A}$, which in its own forks F. Additionally, the time taken by $\mathcal{A}$ for computing the signature and RO queries with additional operations $2(n_s + 2\pi n_h + 1)t_{exp}$ + $\mathcal{O}(\pi n_h + n_s + 1)$. □

Chapter *6*

Conclusions

6.1 Results Summary

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

This makes our protocol better in different ways than other existing protocols. *Table 6.1* illustrates these advantages.

6.2 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}

Table 6.1: Advantages over other 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**

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 requires a single exponentiation computation. The additional misbehaviour detection phase for detecting misbehaviour in one partial signature also requires only one such computation.

6.3 Usage

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 in $t - 1$ corruptions. In addition to faster and asynchronous signing, our protocol is especially effective in the following cases:

- Shared ownership: Shared ownership on the blockchain allows multiple sets of people can decide how to utilise a shared digital asset. Our protocol enables individuals to vote on their decision without waiting for the others to be available.
- Enhanced security: Current threshold security protocols require the quorum of shares to be online at the same time. If a single party wants to distributed their shares in different locations for enhanced security, then this is easier to do in a non-interactive protocol like ours.

6.4 Future Work

Our work is just another result in a long series on work on blockchains. Apart from improving on our result by reducing the communication required in **PreNonceGen** phase, we are also working on alternate techniques to achieve secure asynchronous threshold signing.

Related Publications

- Snehil Joshi, Durgesh Pandey and Kannan Srinathan. "ATSSIA: Asynchronous truly-threshold Schnorr signing for inconsistent availability." Information Security and Cryptology–ICISC 2021: 24th International Conference, Seoul, South Korea, December 1–3, 2021, Revised Selected Papers. Cham: Springer International Publishing, 2022.

  DOI: 10.1007/978-3-031-08896-4_4
  Link: https://link.springer.com/chapter/10.1007/978-3-031-08896-4_4

Bibliography

[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.

[31] Lamport, Leslie, Robert Shostak, and Marshall Pease. “The Byzantine generals problem.” Concurrency: the Works of Leslie Lamport. 2019. 203-226.

[32] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. “Impossibility of distributed consensus with one faulty process.” Journal of the ACM (JACM) 32.2 (1985): 374-382.

[33] Rabin, Michael O. “Randomized byzantine generals.” 24th annual symposium on foundations of computer science (sfcs 1983). IEEE, 1983.

[34] Toueg, Sam. “Randomized byzantine agreements.” Proceedings of the third annual ACM symposium on Principles of distributed computing. 1984.

[35] Bracha, Gabriel. “An O (log n) expected rounds randomized byzantine generals protocol.” Journal of the ACM (JACM) 34.4 (1987): 910-920.

[36] Bracha, Gabriel. “Asynchronous Byzantine agreement protocols.” Information and Computation 75.2 (1987): 130-143.

[37] Fischer, Michael J. “The consensus problem in unreliable distributed systems (a brief survey).” International conference on fundamentals of computation theory. Springer, Berlin, Heidelberg, 1983.

[38] Jakobsson, Markus, and Ari Juels. “Proofs of work and bread pudding protocols.” Secure information networks. Springer, Boston, MA, 1999. 258-272.

[39] Benaloh, Josh Cohen. “Secret sharing homomorphisms: Keeping shares of a secret secret.” Conference on the theory and application of cryptographic techniques. Springer, Berlin, Heidelberg, 1986.

[40] Blakley, George Robert. “Safeguarding cryptographic keys.” Managing Requirements Knowledge, International Workshop on. IEEE Computer Society, 1979.

[41] Asmuth, Charles, and John Bloom. “A modular approach to key safeguarding.” IEEE transactions on information theory 29.2 (1983): 208-210.

[42] Mignotte, Maurice. “How to share a secret.” Workshop on cryptography. Springer, Berlin, Heidelberg, 1982.

[43] Katz, Jonathan. “Digital signatures”. Springer Science and Business Media, 2010.

[44] Wagner, David. “A generalized birthday problem.” Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2002.

[45] Blum, Manuel. “Coin flipping by telephone: A protocol for solving impossible problems.” Advances in Cryptology-A Report on CRYPTO ’81 (1982).

[46] Even, Shimon, Oded Goldreich, and Abraham Lempel. “A randomized protocol for signing contracts.” Communications of the ACM 28.6 (1985): 637-647.

[47] Shamir, Adi, Ronald L. Rivest, and Leonard M. Adleman. Mental poker. Springer US, 1981.

[48] Brassard, Gilles, David Chaum, and Claude Crépeau. “Minimum disclosure proofs of knowledge.” Journal of Computer and System Sciences 37.2 (1988): 156-189.

[49] Goldreich, Oded, Silvio Micali, and Avi Wigderson. “Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems.” Journal of the ACM (JACM) 38.3 (1991): 690-728.

[50] Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof systems.” SIAM Journal on computing 18.1 (1989): 186-208.

[51] Goldreich, Oded, and Yair Oren. “Definitions and properties of zero-knowledge proof systems.” Journal of Cryptology 7.1 (1994): 1-32.

[52] Diffie, Whitfield, and Martin E. Hellman. “New directions in cryptography.” Secure communications and asymmetric cryptosystems. Routledge, 2019. 143-180.

[53] Rivest, Ronald L., Adi Shamir, and Leonard M. Adleman. “A method for obtaining digital signatures and public key cryptosystems.” Secure communications and asymmetric cryptosystems. Routledge, 2019. 217-239.

[54] Ristenpart, Thomas, and Scott Yilek. “The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks.” Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26. Springer Berlin Heidelberg, 2007.