PROBA: PRIVACY-PRESERVING, ROBUST AND ACCESSIBLE BLOCKCHAIN-POWERED HELIOS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERMİN KOCAMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

JANUARY 2024

Approval of the thesis:

PROBA: PRIVACY-PRESERVING, ROBUST AND ACCESSIBLE BLOCKCHAIN-POWERED HELIOS

submitted by **SERMİN KOCAMAN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ayşe Sevtap Selçuk Kestel
Dean, Graduate School of **Applied Mathematics** _____

Assoc. Prof. Dr. Oğuz Yayla
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Ali Doğanaksoy
Supervisor, **Mathematics, METU** _____

Assoc. Prof. Dr. Fatih Sulak
Co-supervisor, **Mathematics, Atılım University** _____

Examining Committee Members:

Prof. Dr. Zülfükar Saygı
Department of Mathematics, TOBB ETU _____

Assoc. Prof. Dr. Ali Doğanaksoy
Department of Mathematics, METU _____

Assoc. Prof. Dr. Oğuz Yayla
Department of Cryptography, METU _____

Assist. Prof. Dr. Buket Özkaya
Department of Cryptography, METU _____

Assist. Prof. Dr. Köksal Muş
Electrical and Computer Engineering, WPI _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SERMİN KOCAMAN

Signature :

ABSTRACT

PROBA: PRIVACY-PRESERVING, ROBUST AND ACCESSIBLE BLOCKCHAIN-POWERED HELIOS

KOCAMAN, SERMİN
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Ali Doğanaksoy
Co-Supervisor : Assoc. Prof. Dr. Fatih Sulak

January 2024, 133 pages

Helios is the first web-based and open-audit voting system. The open-audit feature allows anyone to track the voting process, thus providing easy verifiability in all stages of the elections. Despite many advantages, Helios has a few weaknesses due to its reliance on a centralized server, such as modifying data through unauthorized access or making the server inaccessible. A subsequent work called blockchain-powered Helios is proposed to overcome these weaknesses. This system replaced the centralized server with decentralized servers using the blockchain and a decentralized storage protocol. Although this novelty eliminates Helios' centralized weaknesses, it creates some new problems, thereby causing security weaknesses. These are the misbehavior in the wallet authorization procedure, the linkability in the wallet authorization procedure, and the high cost of transactions. In this thesis, an improved version of the blockchain-powered Helios system, named Proba, is presented. The system is redesigned to provide privacy, robustness, and accessibility in the election. Proba utilizes a novel threshold issuance-anonymous credentials that break the link between the voters and their wallets. Also, the threshold version mitigates the misbehavior of election authorities in wallet authorization. Additionally, Proba leverages a consortium blockchain that provides cost-effective election solutions. In terms of security, the system's formal security concerning specific election requirements is demonstrated through game-based reduction proofs. The performance analysis of

Proba shows that the usage of threshold-issuance anonymous credentials does not have a critical cost for the election phase; on the contrary, it mitigates the smart contract storage cost.

As an additional work on the design of Proba, this thesis proposes enhanced wallet key protection protocols for general blockchain-based I-voting systems. Within the blockchain, transactions must be signed using the wallet secret (signing) key. Thus, the voter's right to send transactions on the blockchain will depend on the security of this single key. In most of these systems, voters can register one wallet address, but in the case of a stolen key, voters are unable to send their votes as a transaction to the blockchain. Although offering voters the opportunity to register their new address is considered a solution in such cases, this creates an avenue for the attacker by forcing them to unregister their existing address. Hence, to prevent such circumstances, rather than permitting the enrollment of new addresses, it is necessary to use enhanced cryptographic measures that can be implemented to protect the existing wallet secret key. The logical precaution is to split the secret key into the different devices of the voter, but in this case, efficient threshold signing protocols must be developed. In this thesis, efficient threshold signing protocols are proposed, specifically focusing on a two-party elliptic curve digital signature algorithm (ECDSA), and a flexible hierarchical threshold signature scheme (FlexHi). In the former, voters split their wallet secret keys across two distinct devices, such as a laptop and a tablet, and then employ two keys to execute a signature on a transaction with our proposed two-party ECDSA protocol. This protocol provides the most optimal offline phase for a two-party ECDSA protocol with such an efficient online phase. In the latter, voters split their wallet secret keys among their various devices, granting different levels of permission to each. During the verification stage of most applications, the transmission of a code to the user's smartphone indicates that the smartphone is deemed a highly significant device, belonging only to the individual. Considering this, voters have the choice to divide their secret keys between their smartphones and other devices. In this scenario, the use of the smartphone, which holds the highest position in the hierarchy, is obligatory during the signing phase. However, secret keys shared with other devices can be used according to the specified threshold value in the system. Current hierarchical threshold schemes incorporate certain ordering criteria and constraints at each level of the hierarchy, which restrict their adaptability. Nevertheless, the proposed FlexHi scheme presents a novel architecture that liberates itself from these limitations and provides flexibility.

Keywords: Helios, Internet Voting, Blockchain, Threshold-issuance Anonymous Credentials, Elliptic Curve Digital Signature Algorithm, Hierarchical Threshold Signature Scheme

ÖZ

PROBA: MAHREMİYETİ KORUYAN, SAĞLAM VE ERİŞİLEBİLİR BLOKZİNCİR DESTEKLİ HELİOS

KOCAMAN, SERMİN
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Ali Doğanaksoy
Ortak Tez Yöneticisi : Doç. Dr. Fatih Sulak

Ocak 2024, 133 sayfa

Helios, ilk web tabanlı ve açık denetimli oylama sistemidir. Açık denetim özelliği, herkesin oylama sürecini takip etmesine olanak tanımaktadır ve böylece seçimlerin tüm aşamalarında kolay doğrulama sağlamaktadır. Helios'un birçok avantajına rağmen, merkezi bir sunucuya bağımlı olması nedeniyle yetkisiz erişim yoluyla verileri değiştirmek veya sunucuyu erişilemez kılmak gibi birkaç zayıflığı mevcuttur. Bu zayıflıkların üstesinden gelmek için blokzincir destekli Helios adlı sonraki bir çalışma önerilmiştir. Bu sistem, merkezileştirilmiş sunucuyu, blokzincir ve merkezi olmayan bir depolama protokolü kullanarak merkezi olmayan sunucularla değiştirmiştir. Bu yenilik Helios'un merkezileştirilmiş zayıflıklarını ortadan kaldırsa da, bazı yeni sorunlar oluşturmakta ve bu nedenle güvenlik zayıflıklarına neden olmaktadır. Bunlar, cüzdan yetkilendirme prosedüründeki hatalı davranış, cüzdan yetkilendirme prosedüründeki bağlantı ve işlem maliyetinin yüksek olmasıdır. Bu tezde, blokzincir destekli Helios'un geliştirilmiş bir versiyonu, Proba, sunulmaktadır. Sistem, seçimde gizlilik, sağlamlık ve erişilebilirlik sağlayacak şekilde yeniden tasarlanmıştır. Proba, seçmenler ve cüzdanları arasındaki bağlantıyı kesen yeni bir eşik verme-anonim kimlik bilgisi kullanmaktadır. Ayrıca eşik sürümü, seçim yetkililerinin cüzdan yetkilendirmesindeki hatalı davranışlarını da azaltmaktadır. Ek olarak Proba, uygun maliyetli seçim çözümleri sağlayan bir konsorsiyum blok zincirinden yararlanmaktadır. Güvenlik açısından, sistemin belirli seçim gereksinimlerine ilişkin biçimsel güvenliği,

oyun tabanlı indirgeme kanıtları aracılığıyla gösterilmektedir. Proba'nın performans analizi, eşik veren anonim kimlik bilgilerinin kullanımının seçim aşaması için kritik bir maliyeti olmadığını göstermektedir; aksine, akıllı sözleşme depolama maliyetini azalttığı görülmektedir.

Proba'nın tasarımına ek bir çalışma olarak bu tez, genel blokzincir tabanlı internet oylama sistemleri için geliştirilmiş cüzdan anahtarı koruma protokolleri önermektedir. Blokzincir içerisinde işlemler, cüzdanın gizli (imzalama) anahtarı kullanılarak imzalanmaktadır. Dolayısıyla seçmenin blokzincir üzerinde işlem gönderme hakkı bu tek anahtarın güvenliğine bağlı olmaktadır. Bu sistemlerin çoğunda seçmenler bir cüzdan adresini kaydedebilmekte ancak anahtarın çalınması durumunda seçmenler oylarını bir işlem olarak blok zincirine gönderememektedir. Her ne kadar seçmenlere yeni adreslerini kaydettirme imkanı sunmak bu gibi durumlarda bir çözüm olarak görülse de, bu durum saldırganı mevcut adresinin kaydını silmeye zorlayarak, kendisine bir yol açmaktadır. Bu nedenle, bu gibi durumları önlemek için yeni adreslerin kaydedilmesine izin vermek yerine mevcut cüzdan gizli anahtarını korumak için uygulanabilecek gelişmiş kriptografik önlemlerin kullanılması gerekmektedir. Uygun olan önlem, gizli anahtarı seçmenin farklı cihazlarına bölmektir ancak bu durumda etkili eşik imzalama protokollerinin geliştirilmesi gerekmektedir. Bu tezde, özellikle iki taraflı eliptik eğri dijital imza algoritmasına (ECDSA) ve esnek bir hiyerarşik eşik imza şemasına (FlexHi) odaklanan verimli eşik imzalama protokolleri önerilmektedir. İlkinde, seçmenler cüzdan gizli anahtarlarını dizüstü bilgisayar ve tablet gibi iki farklı cihaza bölmekte ve ardından önerdiğimiz iki taraflı ECDSA protokolüyle işlem üzerinde imzayı yürütmek için iki anahtarı kullanmaktadır. Bu protokol, böylesine verimli bir çevrimiçi aşamaya sahip iki taraflı bir ECDSA protokolü için en optimum çevrimdışı aşamayı sunmaktadır. İkincisinde, seçmenler cüzdan gizli anahtarlarını çeşitli cihazları arasında paylaştırarak her birine farklı düzeylerde izinler vermektedir. Çoğu uygulamanın doğrulama aşamasında kullanıcının akıllı telefonuna bir kod iletmesi, akıllı telefonun yalnızca kişiye ait, son derece önemli bir cihaz olarak görüldüğünü göstermektedir. Bunu göz önünde bulundurarak seçmenler gizli anahtarlarını akıllı telefonları ve diğer cihazları arasında paylaşma seçeneğine sahiptir. Bu senaryoda hiyerarşide en üst sırada yer alan akıllı telefonun imza aşamasında kullanılması zorunlu hale gelmektedir. Ancak diğer cihazlarla paylaşılan gizli anahtarlar sistemde belirlenen eşik değerine göre kullanılabilmektedir. Mevcut hiyerarşik eşik şemaları, hiyerarşinin her seviyesinde uyarlanabilirliği kısıtlayan belirli sıralama kriterlerini ve kısıtlamaları içermektedir. Fakat önerilen FlexHi şeması, kendisini bu sınırlamalardan kurtaran ve esneklik sağlayan yeni bir mimari sunmaktadır.

Anahtar Kelimeler: Helios, Internet Oylama, Blokzincir, Eşik Verme Anonim Kimlik Bilgileri, Eliptik Eğri Dijital İmzalama Algoritması, Hiyerarşik Eşik İmzalama Şeması

To my family

ACKNOWLEDGMENTS

Firstly, I would like to express my immense gratitude to my thesis advisor Assoc. Prof. Dr. Ali Doğanaksoy who introduced me to this magical world of cryptography and without whom all this could never have happened. He guided me through work in this field and always encouraged me whenever necessary.

I would like to give special thanks to my thesis co-advisor Assoc. Prof. Dr. Fatih Sulak for the time he devoted to me and his enthusiastic encouragement.

I would like to deliver my appreciation to Assoc. Prof. Dr. Oğuz Yayla for the significant amount of time he devoted to me for formal security-proof concepts.

I would also like to convey my respects and thanks to Assist. Prof. Dr. Köksal Muş and Assist. Prof. Dr. Yarkın Doröz for our endless study discussions and the questions they answered sensitively.

I would also like to thank to Dr. Younes Talibi Alaoui for his valuable advice in my studies.

I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting my TÜBİTAK 2244 doctoral program for this research.

Lastly, and most importantly, I would like to thank my family for their support throughout this study and my entire life. I greatly appreciate my husband for his endless encouragement during my journey on my studies and for being the light of my life. Also, I would like to express my gratitude to my mother, my father, and my brother for their support and unconditional love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

I-voting	Internet Voting
E2E-V	End-To-End Verifiability
PBB	Public Bulletin Board
IPFS	InterPlatenary File System
ECDSA	Elliptic Curve Digital Signature Algorithm
ZKP	Zero-Knowledge Proof
NIZKP	Non-interactive Zero-Knowledge Proof
EA	Election Administrator
BPS	Ballot Preparation System
u-ID	Unique Voter ID
LWE	Learning With Errors
DoS	Denial-of-Service
P2P	Peer-To-Peer
TTP	Third Trusted Party
DS	Digital Signature
PPT	Probabilistic Polynomial Time
EUF-CMA	Existential Unforgeability Under Chosen Message Attack
HE	Homomorphic Encryption
IND-CPA	Indistinguishability Against Chosen Plaintext Attack
TIAC	Threshold-Issuance Anonymous Credential
SC	Smart Contract
PoK	Proof of Knowledge
VV	Vote Validity Proof
VS	Vote Sum Proof
Tx	Transaction
CD	Correct Decryption Proof
PoA	Proof of Authority
IBFT	Istanbul Byzantine Fault Tolerance

MtA	Multiplicative-to-Additive
MPC	Multi-party computation
OT	Oblivious Transfer
HTSS	Hierarchical Threshold Signature Scheme
LHR	Linear Homogeneous Recurrence
FROST	Flexible Round-Optimized Schnorr Threshold
SA	Signature Aggregator

CHAPTER 1

INTRODUCTION

An electronic voting system is a method of voting that uses electronic technology to process election data as digital information [6]. In general, electronic voting systems can be divided into two categories: Polling Place Voting and Internet Voting (I-voting). In the former category, voters physically attend a designated polling place and utilize electronic devices to enter their voting choices by pressing buttons. In contrast, the latter option involves voters remotely casting their ballot utilizing computers connected to the internet. I-voting systems provide great accessibility for voters regardless of their locations and allow for a quick evaluation of elections through the implementation of robust algorithms. Also, they reduce the risk of invalid votes ensuring the correctly casted votes. Due to these notable advantages, I-voting is widely preferred in many elections.

I-voting systems should provide the same level of security and confidence as traditional voting [85]. Thus, some basic requirements must be satisfied to gain the trust of the voter. Within the I-voting literature, various voting requirements are established based on potential threats. These requirements have many definitions, as addressed in [147], and can be itemized as follows:

- **Eligibility** Eligibility means voting only by those who are legally allowed to [49]. This means all voters must undergo a secure authorization process before being granted access to participate in the election, and any unauthorized voters will be denied to cast their vote.

- **Privacy.** Privacy is the removal of the link between the voter and the vote

[49]. Since the election systems conceal the vote and the voter's relationship for privacy, this requirement ensures that the voter's choice is anonymous.

- **Uniqueness.** Uniqueness is counting only one vote per voter during the vote-counting phase [31]. Thus, regardless of whether a voter submits multiple votes, each voter possesses equal voting privileges, with the condition that only one vote is considered valid according to the predetermined counting regulations.

- **Fairness.** Fairness means no intermediate result before the election is over [49]. Due to this circumstance, voters who have not yet cast their ballots will vote independently, without being influenced by the result.

- **Robustness.** Robustness refers to the system's ability to resist disruption by any malicious entities [31]. Thus, the system should be designed to withstand various scenarios in which some voters or authorities misbehave in the election. To ensure its resilience, the system must be capable of addressing both external attacks that exploit vulnerabilities in the protocol, such as targeting the operating system or network, as well as internal attacks that involve the misuse of authority or manipulation of voters by malicious entities [87].

- **Integrity** It means the votes cannot be altered, forged, or deleted without detection [142]. Protecting election integrity ensures election result accuracy.

- **Transparency** It means all election phases must be transparent and publicized to the voters [31]. The bulletin board concept can be used to grant access to monitor the election.

- **Receipt-freeness.** Receipt-freeness is the fact that the voters cannot prove to a third party what their vote is, neither during the election nor after the election is over [19]. This desired requirement ensures that an attacker cannot determine the voter's vote, and it prevents vote-selling or buying due to the lack of proof. However, it is important to note that providing receipt-freeness does not necessarily guarantee coercion-resistance, as attackers may still coerce voters into obtaining their private keys and voting on their behalf [66].

- **Coercion-resistance.** Coercion-resistance is the inability of anyone to compel the voter to cast a particular vote or to deter them from voting [66]. This requirement empowers voters to cast their ballots freely, deceiving potential adversaries into complying with their demands.

- **End-to-End Verifiability.** End-to-End Verifiability (E2E-V) allows each voter to check the election phases without requiring trust in election software, hardware, or officials [18]. Being E2E-V encompasses cast-as-intended, recorded-as-cast, and tallied-as-recorded features. In the cast-as-intended feature, voters can verify that their encrypted vote correctly reflects their vote before casting it. In the recorded-as-cast feature, voters can check whether their encrypted votes are accurately captured. In the tallied-as-recorded feature, anyone can verify that all the published encrypted votes are correctly included in the tally. While cast-as-intended and recorded-as-cast links with individual verifiability that allows voters to verify their votes, tallied-as-recorded links with universal verifiability that allows anyone to verify that all valid votes are included in the election result [9].

Some of these requirements may seem contradictory in I-voting systems due to conflicting interests among entities and the absence of physical premises [98]. For example, while individual verifiability allows voters to check their vote, the receipts, such as vote encryption keys, and randomness selected by voters may allow a malicious party to force voters to disclose their votes [35]. Another controversial example is the need to protect a voter's privacy while checking their identity in I-voting systems [98]. Given these conflicting characteristics, the prioritization of system-specific requirements is crucial in the design of I-voting systems.

1.1 Motivation

If I-voting systems are analyzed, it can be seen that the web-based and open-audit Helios voting system [2] is the most popular and promising system. Distinguishing from other voting systems with its transparent and easy verifiability feature, Helios has been used for different real-world elections. University president elections at the

University Catholique de Louvain [3], board member elections at the International Cryptographic Research Association [58], general elections at the Computer Machinery Association [131], and council elections at the Princeton University [144] are some examples of it. Thus, this system needs to be examined when creating an online trustworthy election. Helios system has several strengths that make it one of the best electronic voting systems, some of which can be mentioned as follows:

- Helios provides E2E-V that satisfies the strong transparency of the voting phases [105]. While the voter can audit cast-as-intended and recorded-as-cast conditions, any observer can audit tallied-as-recorded conditions.

- In Helios, as in other voting systems, there is a vote tracking number given in return for the votes sent to the server after they are encrypted in the browser. However, in contrast to previous voting systems, this tracker is not a randomly generated value, but rather the hash of the encrypted vote. The smart ballot tracker utilizes encryption to ensure that the software remains unaltered [67].

- Helios ensures the confidentiality of the votes (since version 2.0) using a distributed encryption scheme. Therefore, no single entity can decrypt individual votes, even in the case of a manipulated entity/server [105].

- Any observer can control the ballot preparation mechanism in Helios because ballot preparation/encryption and ballot casting are designated as two separate steps. Thus, ballot encryption can be viewed before authentication for ballot casting [67].

- Helios proves its security to top experts with open-source codes. By examining the codes, the security of Helios can be easily analyzed [67].

- Helios doesn't compel the voter to use any dedicated hardware or install any specific program because of the web-based feature [105].

- Helios ballot encryption is done using JavaScript and thus, after downloading the necessary information, the voter can disconnect the computer from the Internet and reconnect to the Internet to cast the ballot. So, internet connectivity attacks are useless [119].

- Helios achieves ballot secrecy even during the counting phase; the votes remain encrypted [112].

While Helios has many practical benefits, Helios users need to trust a centralized election server for their stored data. The essential component to publicly verify the cryptographic proofs in Helios is the Public Bulletin Board (PBB) which is used to make the election information public in an append-only, consistent, and verifiable way [65]. However, PBB reads the data from the central PostgreSQL database in Helios. Fortunately, blockchain technology can come to the rescue of trust assumption in Helios with its decentralized structure and removes the need for a secure, centralized server to store election data [70]. Thus, distributed technology is a perfectly suitable alternative to a PBB.

The overall structure of the blockchain enhances the I-voting systems [135]. Firstly, the presence of independent blockchain nodes avoids single points of failure, thereby availability which is the aspect of robustness [125] in the election phase is satisfied. Secondly, the utilization of immutable transactions with hash functions guarantees the integrity of recorded election data. Thirdly, blockchain observability fosters transparency in the election process. Furthermore, the openness of the blockchain enables universal verification of the election [4]. Considering these characteristics, numerous I-voting systems have begun to leverage blockchain technology, as analyzed in [146, 7, 1]. Moreover, various companies have developed fundamental I-voting projects, such as Bitcoin-based Follow My Vote [1] and Agora [2], Ethereum-based Horizon State [3], Hyperledger Fabric-based Luxoft [4], and private blockchain-based Polyas [5]. More information about these projects and their comparisons can be found in [62, 104].

The benefits of incorporating blockchain technology into the I-voting system were also assessed in the Helios system. As a result of this evaluation, the blockchain-powered Helios system was proposed in 2018 [106]. The Blockchain-powered Helios system uses homomorphic encryption like the traditional Helios system, but un-

[1] https://followmyvote.com/
[2] https://www.agora.vote/
[3] https://horizonstate.com/
[4] https://www.luxoft.com/
[5] https://www.polyas.com/

like the Helios system, it integrates blockchain technology. The Blockchain-powered Helios system replaces the centralized broadcast channel, known as PBB, with the decentralized storage protocol called the Interplatenary file system (IPFS), and employs an Ethereum-based smart contract to store and make Helios' audit data public. These modifications enable the decentralized servers to host the open-audit functionality rather than relying on a centralized server in the PBB. Thus immutable data and replicated storage on multiple servers eliminate data modification through unauthorized access, known as data tampering, and inaccessible server status, known as denial of service (DoS) attacks.

Although the advantages of blockchain improve the integrity of Helios, Blockchain-powered Helios introduces new weaknesses related to integrating blockchain structures into the voting system. These weaknesses can be examined as follows:

1. *Misbehavior in wallet authorization.* In the system, the contract owner uploads each voter's wallet address to the smart contract for authorization. As stated in the discussion section of the system, since trust is based on the contract owner, there can be a single point of failure. In this case, if the contract owner doesn't behave as required in wallet authorization, he corrupts the system; thereby, the system fails to satisfy the robustness requirement in pre-election.

2. *Linkability in wallet authorization.* In the system, giving eligibility to the wallet while maintaining privacy is an explicit problem. In the details of the pre-election, the contract owner collects voters' wallet addresses with their authentication information. Once the authentication information is confirmed, the wallet address is inserted into the smart contract, thereby granting authorization. Due to the authentication step in wallet authorization, the contract owner is aware of the connection between the vote and the voter who cast it.

3. *High-cost in transaction.* As stated in the system, Blockchain-powered Helios expects every voter to pay a transaction fee to cast their vote on the blockchain. However, this high transaction fee on the Ethereum blockchain acts as a barrier to voters' participation in the election system and, therefore, to the accessibility of the system.

Apart from these weaknesses, in blockchain-powered Helios, each voter is allowed to register a single wallet address, but if the secret key of that wallet is stolen, the voter cannot participate to the election. Thus, the system lacks advanced measures to counteract the theft of the wallet secret key. However, this is a common problem among most blockchain-based I-voting systems, thus we will analyze it in the context of general blockchain-based I-voting systems. In such scenarios, giving the voter the right to change their address creates an avenue for attackers by forcing the voter to unregister their address. Hence, rather than providing an opportunity to re-register the wallet address, it is imperative to incorporate certain cryptographic techniques to ensure the safeguarding of keys.

In sum, for these weaknesses, we analyzed the security flaws of Blockchain-powered Helios and redesigned it to satisfy privacy, robustness, and accessibility. We also introduced improvements to protocols concerning efficient threshold signatures that can be applied in potential scenarios to ensure the security of wallet keys.

1.2 Contributions

This thesis centers its attention on the development of a novel I-voting system construction. A brief description regarding the contributions of the thesis is presented in Figure 1.1. In system construction, we focused on system design and enhanced wallet key protection measures.

System Design. In the I-voting system design, systems can be collected under five main categories based on the employed cryptographic primitive, namely: homomorphic encryption-based, blockchain-based, blind signature-based, mix-net-based, and post-quantum-based. The advantages and disadvantages of each primitive necessitate the consideration of these factors in the design of the system.

As an initial step, we analyzed the homomorphic encryption-based Helios voting system due to its widespread usage and easy verifiability property. Since the reliance of Helios' users' trust is established on a centralized server, we continued with the decentralized version of Helios, known as Blockchain-powered Helios. While the incorporation of blockchain technology in this version of Helios successfully resolves

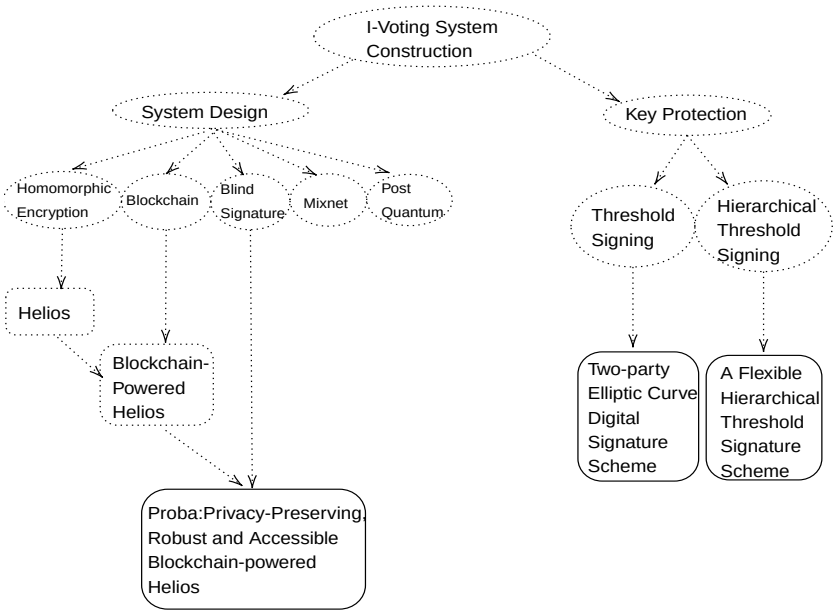

Figure 1.1: Summary of the Thesis Contribution

the issue of a single point of failure that stems from centralized servers, it introduces certain new weaknesses. These primary weaknesses encompass inherent weaknesses that necessitate consideration in the design of the I-voting system. These weaknesses motivated this thesis work. In this thesis, the previously mentioned weaknesses of Blockchain-powered Helios are eliminated, and thus redesigned the system, named Proba, meets privacy-preserving, robustness, and accessibility properties. The methods of Proba voting system to eliminate the weaknesses of Blockchain-powered Helios can be itemized as follows:

1. *Proba eliminates misbehavior in wallet authorization.* To achieve this, Proba distributes the power of wallet authorization to different entities. Within the pre-election process of wallet authorization in Proba, multiple election administrators use a threshold issuance-anonymous credential scheme to sign the wallet public key of an eligible voter. This signed value, in a randomized way, declares that the corresponding wallet address is eligible without the need for the contract owner to upload eligible addresses to smart contracts. In this case, the trust is not based on one authority, thereby robust the election system. Furthermore, by preserving the anonymous credential within the IPFS system, any individual can verify that each casted vote is indeed attributable to an eligible voter.

2. *Proba prevents linkability in wallet authorization.* Proba employs blindness in threshold signature on the wallet public key to prevent linkability between the authentication step and the wallet authorization step. Thus, it helps to preserve the voter's privacy, and nobody knows the link between voters and their wallet, thereby their votes anymore.

3. *Proba reduces the cost of transactions.* Proba uses a consortium blockchain to make the system cost-effective. Thus, the fees imposed on voters for election participation are eliminated. In this case, to prevent excessive transaction requests within the context of the cost-effective transaction fee, a timeout requirement on the public key is set. This indicates that upon casting their votes, voters are unable to immediately submit a second vote, thereby necessitating a waiting period for the update of their vote through their same wallet public key.

Enhanced Key Protection Measure. In addition to the system design, this thesis presents enhanced security measures for the protection of wallet secret (signing) keys in general blockchain-based I-voting systems and thus in the Proba voting system. Generally, the wallet addresses are perceived as electronic mail addresses, while their secret keys are seen as passwords. However, unlike passwords, secret keys cannot be changed in the event of a stolen key, and voters in most blockchain-based I-voting systems are granted the ability to register a single wallet address to avoid double voting. In this case, the most logical way is to split the wallet secret key into multiple devices of the same voter and then use them to sign a transaction. In this scenario, the attacker is required to compromise each wallet secret key, which is impractical in the real world. However, to implement this measure, there is a need to improve efficient threshold signature protocols.

In this thesis, we propose two efficient threshold signature protocols, aiming to protect wallet secret keys against theft in blockchain-based I-voting systems. Specifically, we present an efficient secure two-party protocol on the Elliptic Curve Digital Signature Algorithm (ECDSA) that necessitates the collaboration of two devices of the voter to sign a transaction, and also a flexible hierarchical threshold signature protocol that allows distinct levels of permission for the devices of the voter. The utilization of these novel protocols suggested in this thesis within the blockchain-based I-voting

system and their distinctions from existing threshold protocols in the literature can be analyzed as follows:

- *Two-party ECDSA Protocol.* Voters can distribute their secret key in two places, like a laptop and a tablet, as in Figure 1.2. In such a scenario, the transaction necessitates the utilization of both key shares for signing, rendering it impractical for an attacker to compromise both. To achieve this via an efficient signing procedure, voters can use our proposed protocol, the bandwidth-reduced two-party ECDSA, to securely employ and protect wallet keys. In the two-party ECDSA protocol literature, each protocol relies on heavy homomorphic encryption computations or several executions of multiplicative to additive (MtA) function that convert multiplicative secret shares to additive versions. Unlike these protocols, our protocol introduces an efficient protocol by using only a single MtA function. Also, it reduces communication and computation costs for the offline phase of signing, which is the message-independent phase of signing.

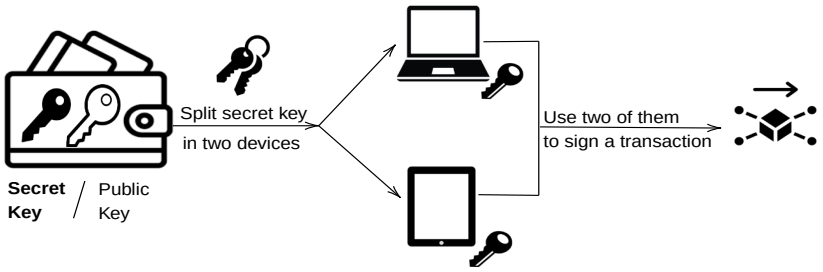

Figure 1.2: Two-Party ECDSA Scenario

- *A Flexible Hierarchical Threshold Signature Protocol.* During the verification phase of some important applications, the transmission of a code to the user's smartphone signifies that the smartphone is regarded as a tool of utmost importance, belonging only to the individual. Considering this, voters can distribute their secret keys between their smartphone, which holds the highest position in the hierarchy, and their laptop and tablet, which hold equal positions in the hierarchy as in Figure 1.3. In this case, it is obligatory to use the secret key share on the smartphone. To achieve this via an efficient signing procedure, voters can use our proposed protocol, a flexible hierarchical threshold signa-

ture scheme (FlexHi). In the literature, many protocols are described on the notion of hierarchical secret sharing, however, existing hierarchical protocols often impose constraints on a threshold value, limiting their flexibility. In contrast to these protocols, our protocol introduces a flexible hierarchical scheme on threshold signatures that employs independent polynomials at each level, thereby eliminating restrictions on threshold values.

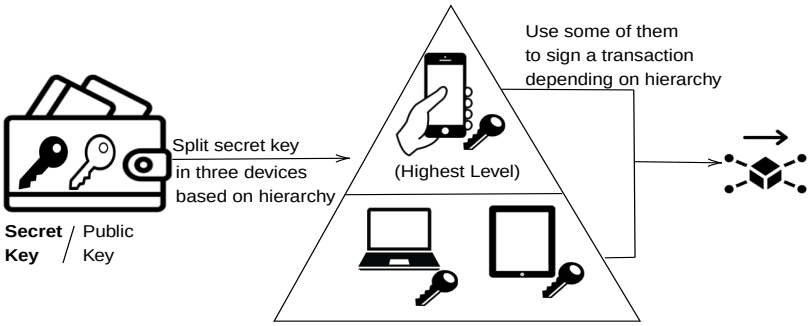

Figure 1.3: Hierarchical Threshold Signature Scheme Scenario

1.3 Organization

The structure of this thesis is as follows: In Chapter 2, an overview is provided on various fundamental aspects, including blockchain technology, I-voting systems, Helios, and the systems inspired by Helios. Chapter 3 presents the architecture of the proposed system, Proba, along with its analysis. Additionally, Chapter 4 introduces two practical threshold signature protocols for protecting the wallet secret key: a two-party protocol for elliptic curve digital signatures and a flexible hierarchical threshold signature scheme. Lastly, Chapter 5 concludes the thesis work.

CHAPTER 2

PRELIMINARIES

2.1 Overview of Blockchain Technology

Blockchain at its core is a peer-to-peer distributed ledger that is cryptographically secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers [11]. In order to gain a clear understanding of what blockchain truly entails, it is necessary to analyze the key terms associated with it:

- **Distributed ledgers.** Distributed ledgers allow for the replication of recorded data and ensure transparency. These ledgers eliminate the possibility of data loss or tampering.

- **Immutability.** The blockchain components like timestamps and hash values oppose changing data and, in turn, provide immutability property.

- **Consensus.** To append the recorded data to the chain, some consensus protocols are applied. Within these protocol rules, the order is maintained easily.

All of these key terms give rise to decentralized technology, namely peer-to-peer technology. Removing trust to a third party, any application can be made still secure with the help of blockchain technology.

To fully comprehend the functioning of blockchain technology, it is essential to delve into its underlying block structures. The blocks consist of the **block header** and the

block body. The 80-byte block header contains the 32-byte Merkle root, the 32-byte hash of the previous block header, the 4-byte Bitcoin version number, the 4-byte timestamp, and the 4-byte difficulty target. The components of the blocks are shown in Figure 2.1. The Merkle root value of the Merkle tree is a single hash value created by hashing each copy of the transaction. The hash value of the previous block is used to calculate the hash value (fingerprint) of each block. Thus, the blocks are chained with the previous block and the integrity of the blocks is strengthened. In addition, the flow in the chain is controlled by controlling the timestamps. On the other hand, the block body consists of transactions, and their transactions are stored in the Merkle tree.

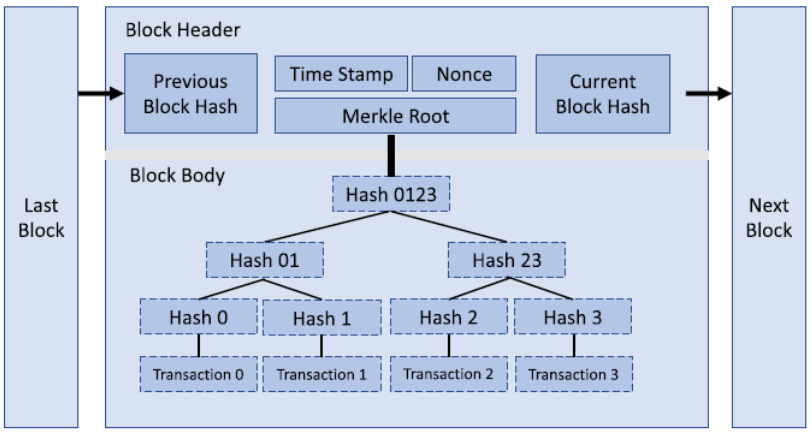

Figure 2.1: Block Components [103]

The subsequent section will elucidate the underlying architecture of the blockchain, thereby facilitating comprehension of the transaction process. Additionally, the meticulous analysis of the transactional stages in Bitcoin allows for a more thorough understanding of the aforementioned process [116].

2.1.1 Blockchain Structures

2.1.1.1 Wallet

Cryptocurrency wallets are digital storage devices used to send or hold any cryptocurrencies. Wallets have two unique components:

- **Wallet keys.** Wallet keys consist of two keys, the signing and verification keys.

While the signing key is used to sign a transaction, the verification key is used to verify the signature on the transaction.

- **Wallet address.** Wallet addresses are the encoded part of the hash of the public verification key and are known to everyone. In the Bitcoin system [94] which is the first application of blockchain, these addresses are produced from public verification keys with SHA-256, RIPEMD160, and base58 encoding [79].

Technically, while the wallet address can be seen as an e-mail address, the private signing key can be seen as the password of the e-mail address. There are two types of wallets: software and hardware [133]. While software wallets are downloadable software programs for computers or phones, hardware wallets are physical vaults that store values on a specially designed hard drive.

2.1.1.2 Hash Reference

A blockchain consists of an ordered list of blocks, and these blocks consist of a set of transactions. Each transaction in the block is identified by its cryptographic hash value, called the hash reference [8].

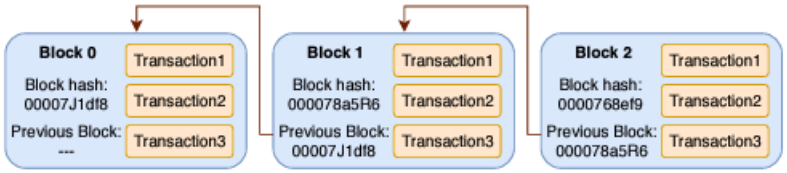

Figure 2.2: Hash Reference [130]

In the blockchain system, each block contains the hash value of the previous block. With this hash reference value, the transactions in the block cannot be changed.

2.1.1.3 Miner

Miners are special nodes that verify the transactions by checking the content of the transaction and the sender [8]. Thus, miners are very important for keeping track of the blockchain. In the mining process, miners add the transaction records to a blockchain's public ledger [130]

2.1.1.4 Consensus

Consensus is a decision-making process that is used to reach an agreement among a group of people [22]. The consensus algorithm ensures trust between the nodes because it bases each block's addition to the chain on it.

Generally, consensus protocols are divided into two categories: probabilistic-finality consensus protocols that are based on voting and absolute-finality consensus protocols that are based on proof [155]. Each consensus model looks at the problems in the blockchain from different perspectives and offers different solutions. Proof-of-Work, Proof-of-Stake, Practical Byzantine Fault Tolerance, Delegated Proof-of-Stake, and Ripple are examples of the main consensus protocols. Their working principles and comparisons can be found in [156, 91].

2.1.1.5 Smart Contract

Smart contracts are executable codes that were first proposed in the 1990s by Nick Szabo [134]. A smart contract triggers the relevant function automatically when its pre-determined requirements written in a computer program are fulfilled.

Many electronic voting applications often require a web server to run the program, but this creates centralization problems. These applications can be executed with smart contracts without using a server so that a single problem on the server does not disrupt the entire system [10]. Therefore, smart contracts are an important technology in voting systems.

2.1.2 Blockchain Types

There are three types of blockchain: public, private, and consortium [96]. Each type has different application scenarios. The comparison of these types of blockchain can be as in Figure 2.3.

Property	Public	Private	Consortium
Management	No Centralized Management	Single organization	Multiple organization
Participants	Permissionless	Permissioned	Permissioned
Centralized	no	Yes	partial
Efficiency	Low	High	High
Consensus Determination	All miners	Organization participating	Selected miners
Transaction duration	Long	Short	Short

Figure 2.3: Comparison of Blockchain Types [135]

2.1.2.1 Public Blockchain

In a public blockchain, everyone can check the transactions and verify them. Also, everyone can participate in the consensus protocol. Bitcoin and Ethereum can be taken as examples of that type of blockchain [96].

2.1.2.2 Private Blockchain

A private blockchain which is also known as permission-based is privately owned by an individual or organization. Also, only authorized nodes can participate in the blockchain network [96].

2.1.2.3 Consortium Blockchain

A consortium blockchain is a hybrid form of public and private blockchains. Consortium blockchain usually has partnerships like business to business and there are predetermined nodes in the blockchain network [96].

2.2 Overview of I-voting Systems

The general I-voting system is composed of four stages: announcement, registration, ballot casting, and tallying [118]. These can be collected under three headings: pre-election, election, and post-election, as in Figure 2.4. The pre-election phase consists of announcements and registration steps. The system parameters, the eligible voters,

and the candidate list are determined in the announcement step. In the registration step, the voters receive or authorize their digital credentials after proving their identity to the authorized administrator. The election phase consists of the ballot-casting step, in which the voter casts a vote with the credential information obtained in the registration. On the other hand, the post-election stage of tallying counts the votes. It should be noted that there is also a verification stage, which serves to control the information by the voter and other participants. Since it can be used in each stage except the announcement, this stage has been excluded from the stages of the election.

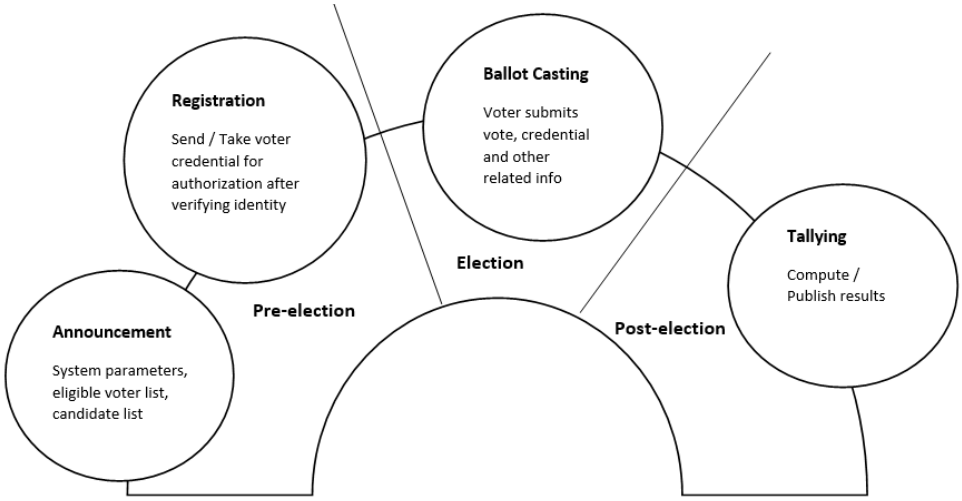

Figure 2.4: Stages of Elections

Although the stages and requirements of I-voting systems are the same, many I-voting systems proposed in the literature have used different primitives. In the next subsections, firstly, the categories of I-voting systems, which are divided according to the used primitives, will be examined, and then the Helios and Helios-inspired noteworthy systems will be analyzed.

2.2.1 Classification of Internet Voting Systems and Cryptographic Primitives Behind

According to the used primitive, I-voting systems can be divided into five main categories: blind signature-based system, mix-net-based system, homomorphic encryption-based system, post-quantum-based system, and blockchain-based system [69]. Using these protocols together or alone at different stages of the election has its advantages

and disadvantages.

2.2.1.1 Blind Signature-Based System

The idea of blind signature was first introduced by [32], and the first blind signature-based e-voting system was created by [49] using this idea. The difference between a blind signature and an electronic signature is that the information in the blind signature is not known by the signer.

In Blind Signature-Based systems, the original data is masked with a randomly chosen value, and then this masked value is signed. By removing the random number, which is used in the masking operation, from the signed masked value, the signature of the original data is obtained. Thus, a blind signature can be seen as a method that provides blindness and untraceability [75]. While blindness ensures that the content of the data is not known by the signer, untraceability ensures that the signer cannot trace those signatures. Also, blind signatures don't require high communication costs to ensure anonymity but require an anonymous channel [98].

2.2.1.2 Mix-Net-Based System

In Mix-net-Based systems, the main idea is to shuffle and encrypt/decrypt the data. This method was first used in Chaum's diagram [34]. Mix nodes, which are the basis of the mix network model, are anonymous servers connected in the network. In the shuffle process, each mix node shuffles the votes from the previous node and sends them to the next node. Since the permutation value here is hidden, the storage order of the votes cannot be predicted over the mix network.

In addition to this mixing process, mix servers apply a re-encryption or decryption process to prevent access to the votes from input and output values comparisons. With this method, the link between the voters and their votes is removed. Also, since each vote is decrypted individually, corrupted votes can be detected in tallying phase without requiring vote validity proofs [23].

Decryption Mix-Net. In these mix-nets, votes are encrypted by the voters as much

as the nodes in Mix-net, using the public key of each mix node in the encryption layer. Each node in the network, after the shuffling process, takes the encrypted vote collection, decrypts one of the encryption layers with its own private key, and transfers the votes to the next node. The main disadvantage of this method is that the voter has to encrypt the vote as much as the nodes in mix-net [110].

Re-Encryption Mix-Net. In these mix-networks, random public-key encryption schemes such as Elgamal and Paillier are used, which allow the votes to be re-encrypted with a random number. Thus, with the shuffling, the votes are re-encrypted and transferred to the next server, and a single decryption step takes place in the decryption of the votes. In this method, the voter only needs to encrypt the vote once. Also, contrary to the decryption mix-nets, the failure of a single server in re-encryption mix-nets will not disrupt the entire voting process [114].

While each mix-net type breaks the link between the voters and their votes, the mix nodes must ensure that mixing was correctly constructed; otherwise, a corrupt mix node deletes or adds fake votes. But proving the correctness of the shuffle in verifiable mix-nets requires complex protocols [98]. Also, due to its complexity, mix-net-based voting systems aren't suitable for large-scale elections [69]. Moreover, linked mix nodes are vulnerable to DDOS attacks [69].

2.2.1.3 Homomorphic Encryption-Based System

In the homomorphic encryption-based systems, which were first used in [40], computations on encrypted votes are done without the need for decrypting any single vote thanks to an algebraic homomorphic property. For partially homomorphic encryption, elections can use additively or multiplicatively homomorphic schemes. The only difference is in tallying phase.

If the election uses one of the additively homomorphic schemes like Paillier [99], and Goldwasser-Micali [56], then decryption on aggregated ciphertexts will result in the sum of the votes.

$$DEC(Enc(m_1) \cdot Enc(m_2)) = DEC(Enc(m_1 + m_2))$$

If the election uses one of the multiplicatively homomorphic schemes like RSA [115] and ElGamal [46], then decryption on aggregated ciphertexts will result in the product of the votes, and the product is then factorized to obtain votes.

$$DEC(Enc(m_1) \cdot Enc(m_2)) = DEC(Enc(m_1 \cdot m_2))$$

In general, homomorphic encryption has two important benefits for the election [40]. Firstly, they serve to remove the link between the voters and their vote because vote counting doesn't require the decryption of votes individually. Secondly, they serve to satisfy the universal verification for elections because anyone can obtain the sum of all votes to compare the results.

The disadvantage of this method is the intensive zero-knowledge proof (ZKP) for the validity of votes [139]. Thus, if an efficient zero-knowledge proof is not used, it will not be suitable in elections. In addition, as the number of candidates increases, the data size increases and the speed of the system slows down, so it is much more efficient for yes/no elections [139].

2.2.1.4 Post-Quantum-Based System

The security of most existing algorithms relies on three complex mathematical problems: the integer factorization problem, the discrete logarithm problem, and the elliptic-curve discrete logarithm problem. Shor [126] discovered in 1994 that integer factorization and discrete logarithm problems can be solved in polynomial time on quantum computers. Some voting systems are designed to use post-quantum algorithms for the threat of quantum computing.

In Post-Quantum I-voting systems, the systems defend themselves against quantum adversaries by choosing quantum-resistant algorithms in election phases [50]. Thus, the systems protect themselves for a long span against the threat of solving mathematically complex problems in polynomial time with quantum computers. However, they require more storage space due to the larger key size [69]. Also, the large size of

the data limits the speed of the transmission [69].

2.2.1.5 Blockchain-Based System

Blockchain technology, whose core idea was first proposed in Bitcoin white paper [95], can be seen as a decentralized digital ledger in a peer-to-peer network. In Blockchain-Based systems, decentralized, immutable, and transparent features provided by the blockchain are utilized. A vote is represented as a transaction that includes the transaction index, timestamp, and hash of the current transaction. When the voter chooses the candidate, this transaction is queued, and the nodes in the network validate the transaction through a consensus algorithm. When the required time and transactions are satisfied, a block is created.

In general, blockchain can be designed in three different ways, a public, private, or consortium blockchain, depending on the requirements [135]. The ownership of the blockchain and participant permission on the blockchain will determine the type of blockchain. Briefly, blockchain technology provides immutable vote records with linked hash values and offers transparency. However, it faces authentication, anonymity, and scalability problems. The authentication problem is related to the fact that eligible voters' real-life identities must be bound to wallet addresses to have the right to vote [61]. However, this link which is required to be able to vote, must also not ruin anonymity during the election. In addition, blockchain with large-scale elections causes scalability problems due to the higher cost and time-consuming transactions [1].

In conclusion, many I-voting systems have been proposed based on the company and academic studies using these basic methods. More information on them and their comparisons can be found in [69], [62]. In general, each method possesses its own set of benefits and drawbacks. Considering the pros and cons of various methods, it is evident that hybrid schemes, which combine two or more methods, are more practical and efficient compared to other systems [69]. The inheritance of advantages from the combined voting approach effectively mitigates the majority of disadvantages present in each separate voting method.

2.3 Helios I-voting System and Its Security Analysis

Helios system has been used in many real-world elections due to its E2E-V property. Being E2E-V encompasses [18]:

1. Cast-as-intended: Voters can verify that their encrypted vote correctly reflects their vote before casting the encrypted vote.
2. Recorded-as-cast: Voters can check that their encrypted votes, which are displayed on the public list, are accurately captured.
3. Tallied-as-recorded: Anyone can verify that all the published encrypted votes are correctly included in the tally.

Helios satisfies E2E-V as follows: ① In Helios, the cast-as-intended phase is achieved by a cast-or-audit technique which is known as Benaloh challenge [16]. In this technique, each voter can choose to either audit her encrypted ballot or submit it. In the case of an audit, voters get proof that the encrypted vote was correctly constituted. In fact, voters get the random value used in encryption, and then, by recomputing the encryption, they can check whether the same encrypted vote is obtained. A party or tool that is independent of the voting server should carry out this recomputation. Once encryption is under control, an audited vote will act as a 'challenge', and the encrypted vote must be generated again to maintain vote confidentiality before submission. The key to the success of this technique is that it is not known which encryptions will be challenged. ② In Helios, the recorded-as-cast phase is achieved by giving receipts to the voters about their encrypted vote. Voters take the hash of their encrypted vote as a receipt and can then check whether the encrypted vote is published correctly in the public list. ③ In Helios, tallied-as-recorded phase is achieved by cryptographic mathematical proofs. With homomorphic encryption used in Helios, the tally of the encrypted votes can be done without decrypting them. Thus, it is possible to compare the announced voting results to match the published encrypted cast votes.

The first version of Helios emerged with a mix-net-based technique [2]. In this version, the voter's device encrypts the candidate option and sends this ciphertext to the server with the voter's authentication information. After the voting is finished, Tal-

liers anonymizes the cast ballots using a re-encryption mix-net with verifiable proofs. However, Helios's mix-net-based version is inefficient due to proofs of shuffles. According to their performance measurement for 500 voters with a two-question election, while the shuffle proofs take 3 hours, the verification steps take 4 hours. Thus, to anonymize the ballots without an intensive computation process, Helios migrated from mix-net to homomorphic encryption [3]. Unless the first version of mix-net is specified in the rest of the paper, the Helios system will be taken as homomorphic encryption-based.

2.3.1 Helios System Architecture

2.3.1.1 Entities

Helios consists of three types of participants:

- **Election Administrator (EA)** manages the decisions related to the election.
- **Voter** connects to the election through the link and casts his vote with the given credential.
- **Trustee** is responsible for generating the election encryption key and performing the decryption at the end of the election.

In addition to these participants, Helios has the Helios election server and Ballot Preparation System (BPS). While the Helios election server runs the election with commands, BPS serves as a service on the voter's browser and sends it with its proof of correctness.

2.3.1.2 Helios System Construction

The steps of the Helios voting procedure can be explained in detail below:

1. EA defines the start/end time of the election, election questions, and the email addresses of the Trustees and allowed voters.

2. Trustees generate election key pairs via the link in their email and send the public one to the Helios server.

3. Helios server generates the election public key in a distributed way through Trustees. At this point, hashing public election parameters produces the election fingerprint.

4. When the election is open, vote invitation mail is sent to the voter's e-mail containing a unique voter ID (u-ID), 10-character password, alias, election fingerprint, and election link.

5. The voter selects a candidate and reviews the ballot, then casts an encrypted vote and related ZKP with u-ID & password as a credential. In fact, in the review, the voter's ballot tracker (hash of the encrypted vote) is shown to the voter.

6. After the credential control, cast votes and ZKP from the server's PostgreSQL database published on a publicly available system component called PBB. Also, all voters can see the alias and corresponding ballot tracker to compare the votes.

7. After the election, the server takes encrypted votes and ZKPs then aggregates the encrypted votes whose ZKPs are valid.

8. Each Trustee takes this aggregated encrypted tally and decrypts it with their private key partially. Also, they ensure the correct partial decryption by another ZKP.

9. As a result of these operations, the decryption process is performed due to the use of the homomorphic encryption algorithm, and the results are displayed in the PBB.

The steps of voting followed in the Helios system can be seen in Figure 2.5. In this figure, the steps on the pre-election, election, and post-election, are numbered 1,2,3, respectively, and divided into sub-numbers.

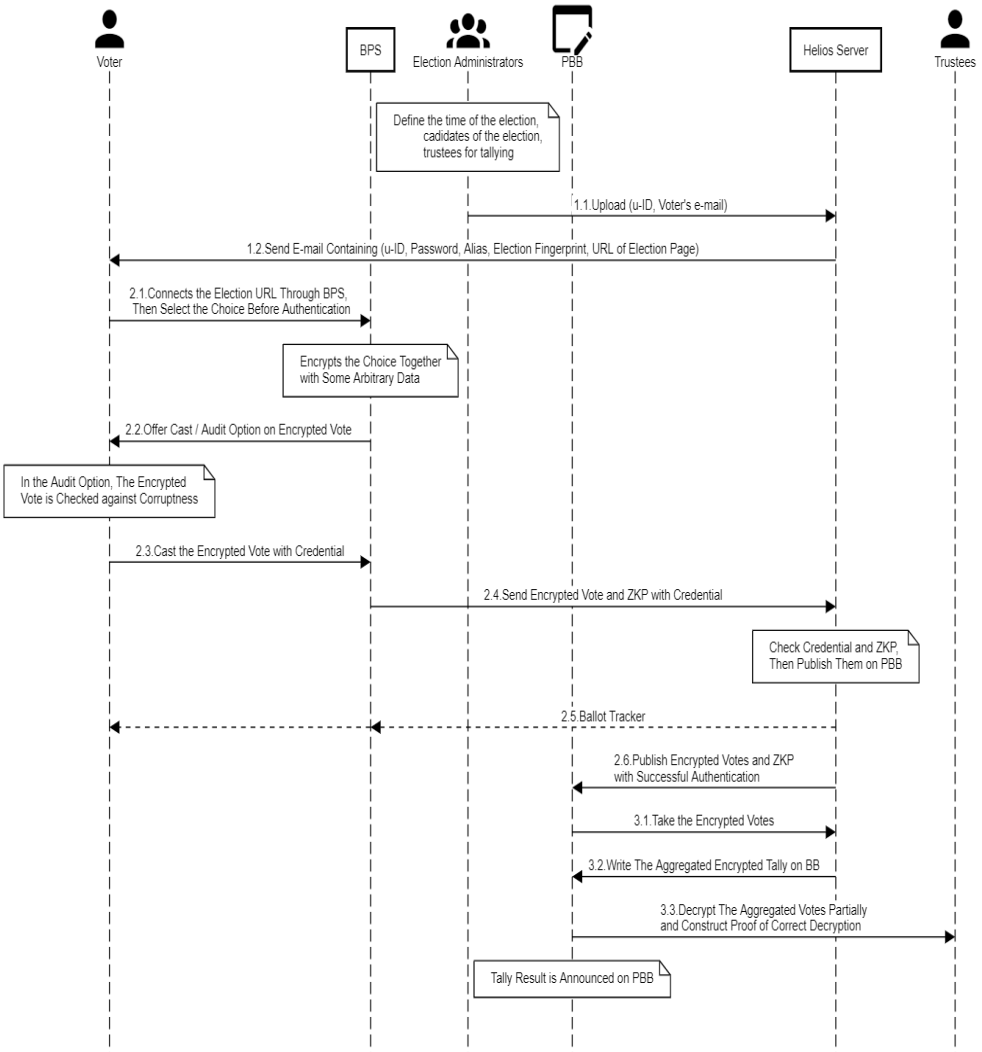

Figure 2.5: Helios Voting Procedure

2.3.2 Security Analysis

According to the following security analysis, it is seen that Helios doesn't satisfy robustness, receipt-freeness, and coercion-resistance.

- *Eligibility.* Helios gives each eligible voter to unique voter id and password, then PBB only accepts the votes that come from the eligible voter's identity. Thus, only eligible voters can vote in the Helios.

- *Privacy.* While the votes are encrypted with homomorphic encryptions, each voter casts this encrypted vote with the given alias in Helios. Since nobody knows the relation between the voters and their aliases, privacy is satisfied.

- *Uniqueness.* PBB in Helios allows a maximum of one vote per identity, thus satisfying the uniqueness.

- *Fairness.* Helios uses distributed key generation to generate an election encryption key, and after the election, each Trustee partially decrypts the value with their secret key. Since nobody knows the master decryption key, anyone cannot decrypt the values alone.

- *Robustness.* When the voter makes his selection, the browser encrypts the voter's choice in the background, and then creates zero-knowledge proof. These are vote range proof to prove that the ciphertext encrypts the allowed candidate values and vote sum proof to prove that only one of the allowed candidates is chosen in the ciphertext. This ensures that malicious voters cannot disrupt the system. However, since the data on PBB is read from a central PostgreSQL database, malicious election authority can cause a single point of failure.

- *Integrity:* Since Helios provides its users to a hash of encrypted votes as a receipt, there cannot be an unauthorized change in the votes. Thus, it satisfies the integrity.

- *Transparency:* Since Helios uses a PBB structure to provide its users to monitor the election, it satisfies transparency.

- *Receipt-freeness.* Since the random number used in vote encryption is chosen by the voter's client, remembering randomness proves how the voter voted; thus, helios is not receipt-free [89].

- *Coercion-resistance.* Helios offers a coerce me button, and as stated in the article, it does not protect against coercion.

- *Verifiability.* Helios offers E2E-V. Giving voters receipts for their encrypted votes achieves the recorded-as-cast feature, whereas the cast-as-intended feature uses the cast-or-audit technique known as the Benaloh challenge. Additionally, homomorphic encryption and cryptographic proofs enable the tallied-as-recorded feature in Helios.

2.4 Noteworthy Helios-Inspired Systems

Several I-voting protocols have been proposed in the literature due to the prominent Helios voting system. However, every Helios-inspired system has its non-trivial problems. In this section, we will give a brief summary of the most noteworthy Helios-inspired systems.

Apollo. Apollo system prevents vote manipulation in homomorphic Helios, introducing two-step vote-casting [51]. Once the voter confirms the vote on PBB, then enters the lock-in code; otherwise, the casting code changes the vote. Thus manipulated votes are determined before tallying, but delivering the codes to the voter through a channel securely poses usability problems.

Zeus. Zeus [143] is based on mix-net-based Helios and uses El Gamal with the re-encryption mix-net technique to anonymize the ballots. To support several different types of ballots, such as multiple choice questions, Zeus adopted a homomorphic encryption implementation of Helios for tallying the ballots partially but not for producing the election result. In that case, a verifiable ballot count can be fed to the election result calculator of any type of election. In the election, the votes are shuffled by Zeus and Trustees' respectively, and each mixer submits proof that they have performed the mixing correctly. Once the election has been closed, the mixed set of re-encrypted ballots is exported to Trustees for partial decryption. Then the result is sent to the Zeus server for final decryption and final election results. A limiting factor in this scheme is that mix-net implementation is not efficient; in an hour, only a few thousand votes can be handled.

Learning With Errors-Based Helios. The other system inspired by Helios is the learning with errors (LWE) based system proposed by Chillotti et al. [36]. Instead of using partially homomorphic encryption, they use an LWE-based fully homomorphic scheme. In that case, they removed the need for vote validity proof on the voter's side. Also, on the authority side, they used publicly verifiable ciphertext trapdoors. However, using fully homomorphic encryption causes efficiency problems for the system. For that reason, there is no implementation of this system.

Voatz. Voatz [123] used private blockchain technology in the system and used face

recognition for authentication. In the system, each voter logs into the system with the face identification application and sends the vote through the application. Meanwhile, a unique anonymous voter ID is generated and kept in this ID along with the vote in the blockchain. A signed receipt is sent to the voter. After the election, two election officials assembled the votes on the blockchain with a pdf file on a USB flash drive and explained the results. In this system, since the authentication phase depends on the face recognition system, it is open to many attacks.

Belenios. Belenios [38], which is built upon Helios, provides eligibility verifiability as an additional feature to prevent ballot stuffing in case of malicious servers. Thus, through a signature mechanism and additional credentials, anyone can check the ballot on the bulletin board. To achieve this, Belenios split the authentication phase into login and password authentication. Each part is managed by a different authority, named the registrar and voting Server. After the registrar sends different signing keys to each voter, he/she transmits the corresponding list of verification keys to the voting server in random order. Also, the voting server sends different passwords to each voter and publishes all verification keys. In the election, each vote is encrypted using an exponential El Gamal algorithm and then signed with a Schnorr signature to prevent any ballot stuffing. The server takes encrypted votes, proofs, signatures, and passwords, thus controlling the values before decryption. In brief, any unauthorized ballot cannot be added to the bulletin board unless both the registrar and the voting server are corrupted to share the secret key of the voter and password. However, the secure distribution of voters' private signing keys to voters is an important problem.

Ordinos. Ordinos [76] also utilize the general structures of the Helios, but unlike Helios, it doesn't reveal the full tally because it is a low privacy situation for very small elections. Thus, Ordinos extends the Helios to support tally-hiding elections in a verifiable way. In a nutshell, voters in Ordinos send homomorphically encrypted votes to the bulletin board. After aggregation on ciphertexts, Trustees run the "greater-than" Multi-part computation protocol [84] to secretly evaluate and publish the result function with its correct evaluation proof. In that case, even Trustees learn nothing beyond the winner of the election. While it is a desirable property for very small elections, Ordinos does not fit well in scenarios where transparency of the election results is a prerequisite, like political elections.

Blockchain-Powered Helios. All centralized Helios-inspired systems suffer from single points of failure. However, the blockchain-powered version of Helios overcomes this weakness [106] by taking advantage of the decentralized structure of blockchains.

Since blockchain technology provides immutable vote records with linked hash values and offers transparency, we analyze the blockchain versions of Helios-inspired systems in detail. Voatz is not clear enough to analyze since many details of the election stages in Voatz are not specified precisely, and these uncertainties are indicated in [63]. Thus we analyze the Blockchain-powered Helios in detail in the next section.

2.5 Blockchain-Powered Helios I-voting System and Its Security Analysis

Despite the strengths of Helios, Helios has a few weaknesses that arise from centralization. However, Blockchain-powered Helios [106] overcome these problems in Helios. Since the PBB in Helios reads the data directly from a central storage system, there can be data tampering [106]. Also, since the access is provided over the internet, the attacker can send unlimited requests and cause to Denial-of-Service (DoS) attack, and thus the server cannot work correctly [119]. Blockchain-powered Helios has changed the centralized PBB structure, in which the vote tracking information is published publicly, and changed the centralized storage system where this information is read. In that system:

- The centralized PBB structure has been replaced with the blockchain.

- The centralized storage system has been replaced with a decentralized peer-to-peer (P2P) storage solution called IPFS.

With that solution, data tampering and DoS attacks are eliminated [78]. Due to IPFS system integration, the hash value in the blockchain is used as an IPFS link that contains encrypted votes and ZKPs; thus, accessing encrypted votes from the hash value becomes easy in the blockchain.

2.5.1 Blockchain-powered Helios System Architecture

2.5.1.1 IPFS

Peer-to-peer (P2P) data networks have emerged to reduce the trust assumption in data storage [41]. These P2P data network generations have gained popularity over a centralized network. Specifically, the most popular P2P system, the InterPlanetary File System (IPFS) [20], is preferred as a storage layer for blockchains.

IPFS offers a content-based distributed file-sharing protocol to increase the web's efficiency and resistance to centralization attacks. Data in IPFS is split into partitions among the peers. From these partitions, a hash graph similar to the Merkle tree is constructed, and the root of this graph gives the original file. To ensure the availability of the files, IPFS also has its incentive layer as Filecoin [77].

IPFS is also used in Blockchain-powered Helios as a distributed storage protocol that complements blockchain. While the election data is kept on the IPFS protocol, their hash value, an IPFS link, is kept on the blockchain. Thus through the IPFS link, anyone can access the original data storage. Thanks to IPFS, two significant advantages are obtained over the blockchain. First, it reduces the blockchain scalability issues caused by big data storage. Second, transaction fees are also reduced due to less data stored on the blockchain.

2.5.1.2 Entities

Blockchain-powered Helios consists of four types of participants:

- **Election Administrator (EA)** creates the smart contract for the election and uploads voter-chosen wallet address to the smart contract via the helios election server.
- **Voter** generates his wallet address and casts his vote as a transaction through his authorized wallet address.
- **Blockchain** is responsible for the publication of the IPFS address of the voter's

vote.

- **IPFS** stores election audit data on its decentralized network.

2.5.1.3 Blockchain-powered Helios System Construction

The steps of the Blockchain-powered Helios voting procedure can be explained in detail below:

1. Before the election, each voter generates their wallet address and sends it to EA with authentication information.

2. EA uploads the Ethereum wallet address of successfully authenticated voters to a smart contract.

3. When the election starts, the voter authenticates with the Helios election server after accessing the link for the election.

4. Voter selects a candidate. Then the voter's encrypted vote is stored in IPFS, and the hash of the file, the IPFS link, is given to the voter as a receipt.

5. After checking, the voter signs this link and send it as a signed transaction to the blockchain.

6. When the data is added to the blockchain, the voter gets the transaction Id, and the blockchain publishes the IPFS links of the encrypted vote that is stored in IPFS.

7. After the election, encrypted votes are read from the IPFS address in the blockchain, and vote results are counted in the same manner as Helios.

The steps of voting followed in the Helios system can be seen in Figure 2.6. In this figure, the steps on the pre-election, election, and post-election, are numbered the 1,2,3, respectively, and divided into sub-numbers.

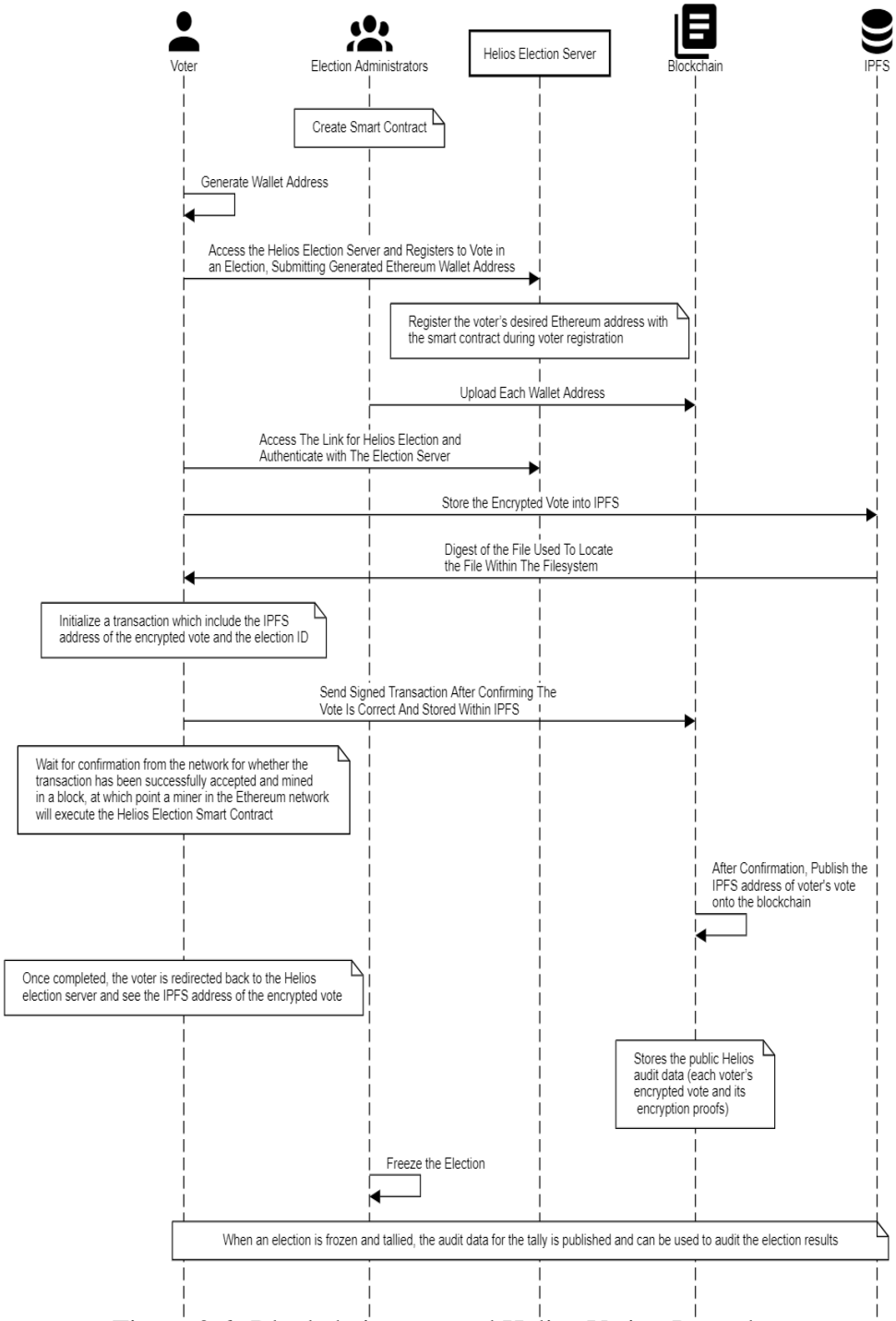

Figure 2.6: Blockchain-powered Helios Voting Procedure

2.5.2 Security Analysis

According to the following security analysis, it is clear that Blockchain-powered Helios does not meet the privacy and robustness features. Also, it doesn't satisfy receipt-freeness and coercion-resistance features, as in Helios.

- *Eligibility.* Since smart contracts accept only votes from eligible wallet addresses, the system satisfies the eligibility.

- *Privacy.* Since EA takes the voter-chosen wallet address with the voter ID, the correlation between the voters and their wallets is known by EA; thus, the system doesn't satisfy privacy.

- *Uniqueness.* In the system, each voter has the right to have a wallet address defined in the system, so only one of the votes coming from this special address is counted during the counting phase.

- *Fairness.* Although the generation of the election key is not clear in the Blockchain-powered Helios, the distributed key generation algorithm can also be used in this system, which is built based on the Helios system, and thus the system also provides the fairness feature.

- *Robustness.* In the system, since EA takes the voter-chosen wallet address and uploads it to the smart contract, he can corrupt the system by not acting properly; thus, robustness is not satisfied.

- *Integrity:* Since Blockchain-powered Helios is based on blockchain technology, encrypted votes are seen as block transactions. Each transaction has a transaction hash/Id, thus there cannot be an unauthorized change in the votes. Thus it satisfies the integrity.

- *Transparency:* Since Blockchain-powered Helios use a public blockchain, everyone has direct access to data and transaction submissions Thus, it satisfies the transparency.

- *Receipt-freeness.* Since each voter takes transaction ID as a blockchain transaction receipt, the system doesn't satisfy receipt-freeness.

- *Coercion-resistance.* The system is based on Helios, and as in Helios, no extra measures have been taken for resistance to coercion.

- *Verifiability.* Blockchain-powered Helios offers E2E-V as in Helios. For a cast-as-intended property, the voter takes the hash of this IPFS file which specifies a link. After controlling this property, the voter signs this hash value and sends it

as a signed transaction in a block. In this system, encrypted votes are stored in IPFS, and their IPFS links are published on the blockchain. Thus, the recorded-as-cast property is satisfied. As with each blockchain transaction, each voter must obtain a transaction ID value to display which block the vote information is in. This also assures tallied-as-recorded property.

To sum up, apart from the advantages provided by the use of blockchain and decentralized storage, some new weaknesses can also be observed in Blockchain-powered Helios. These weaknesses focus on blockchain integration:

1. *Misbehavior in wallet authorization.* EA in Blockchain-powered Helios uploads voter-chosen wallet addresses to the smart contract. As stated in the discussion section of the system, since trust is based on the contract owner, there can be a single point of failure. Thus, the malicious administrator can corrupt the system, and system robustness cannot be met.

2. *Linkability in wallet authorization.* In the system, associating wallets with voters is an explicit challenge. To associate wallets with voters could violate the voters' privacy since the contract owner knows the link between the voter and the vote comes from a specific wallet address. Thus the system doesn't satisfy privacy.

3. *High-cost in transaction.* As stated in the system, Blockchain-powered Helios requires every voter to pay a transaction fee to publish their vote on the blockchain. However, this minimum transaction fee can be a burden for the accessibility of the system.

Also, wallet secret key protection is an important issue for Blockchain-powered Helios, and also for most of blockchain-based I-voting systems. Thus, we will evaluate this issue separately. In the next chapter, mentioned above three weaknesses in Blockchain-powered Helios are eliminated, and a new election system named Proba is proposed. Following that, efficient threshold signing protocols are proposed to mitigate wallet secret key theft scenarios.

CHAPTER 3

PROBA: PRIVACY-PRESERVING, ROBUST AND ACCESSIBLE BLOCKCHAIN-POWERED HELIOS

This chapter presents a novel I-voting system called Proba, which effectively fulfills the requirements of privacy, robustness, and accessibility. After giving the necessary cryptographic blocks to build this system, system construction of the Proba will be given. Additionally, this chapter encompasses the security and performance analysis of the Proba.

3.1 Building Blocks

This section first presents the notion of type III bilinear pairing for the threshold issuance anonymous credential scheme used in Proba. Subsequently, it presents three distinct security assumptions and two indispensable concepts utilized in Proba.

Syntax. For the syntax of algorithms, we use standard notations. If A is an algorithm, then $\mathsf{A}(\mathsf{x})$ is the result of this algorithm on input x. While $\mathsf{A}(\mathsf{x}) \rightarrow \mathsf{y}$ denotes that y is an output of the algorithm A on input x, $\mathsf{x} \leftarrow \mathsf{y}$ denotes the simple assignment process. Also, $\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{Pr}, \mathsf{Vr}$ stand for key generation, signing, encryption, partial decryption, decryption, proof, and verification, respectively. In addition, $\mathcal{DS}$, $\mathcal{HE}$, $\mathcal{ZKP}$, and $\mathcal{TIAC}$ represent digital signature, homomorphic encryption, zero-knowledge proof, and threshold-issuance anonymous credential, respectively.

Definition 1 (Bilinear Pairings [45]). *Let $\mathbb{G}_1$ (additive), $\mathbb{G}_2$ (additive), and $\mathbb{G}_t$ (multiplicative) cyclic groups of prime order p, g_1 be an arbitrary generator of $\mathbb{G}_1$, g_2 be*

an arbitrary generator of $\mathbb{G}_2$. *A bilinear pairing* $\mathsf{e}:\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ *(type III where* $\mathbb{G}_1 \neq \mathbb{G}_2$ *and there is no efficiently computable homomorphic function) is a mapping having the following properties:*

- *Bilinearity: For all* $\mathsf{a}, \mathsf{b} \in \mathsf{Z_p}$, $g_1 \in \mathbb{G}_1$, *and* $g_2 \in \mathbb{G}_2$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$
- *Non-degeneracy:* $e(g_1, g_2) \neq 1$ *for some* $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$

Definition 2. *(Discrete Logarithm (DL) Assumption [26]) Let* $\mathbb{G}$ *be a cyclic group of prime order p, with generator g. DL assumption is that given* g^x *where x is chosen randomly in* $\mathbb{G}$*, finding such an x is difficult to find in the underlying group* $\mathbb{G}$.

Definition 3. *(Decisional Diffie Hellman (DDH) Assumption [26]) Let* $\mathbb{G}$ *be a cyclic group of prime order p, with generator g. DDH assumption is that given the tuple* (g^x, g^y, g^z)*, finding whether* $x \cdot y = z$ *mod p is computationally hard in the underlying group* $\mathbb{G}$.

Definition 4. *(eXternal Diffie-Hellman (XDH) Assumption [25]) Let* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ *be bilinear map with order p with generators* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$. *XDH assumption states that the DDH assumption is intractable in* $\mathbb{G}_1$.

Definition 5. *(Semantic Security (SS) [127]) It ensures that an adversary cannot practically distinguish the encryption of his two messages of choice. Formally, it means any efficient adversary's SS-advantage (*$|Pr[b = b'] - \frac{1}{2}|$*) in the following game is negligible:*

- *The challenger* $\mathcal{C}$ *generates the public/secret key pair (*pk, sk*), then gives* pk *to* $\mathcal{A}$
- *The adversary* $\mathcal{A}$ *chooses two distinct but the same length messages* $\mathsf{m_0}$ *and* $\mathsf{m_1}$*, then sends them to the challenger* $\mathcal{C}$.
- *The challenger* $\mathcal{C}$ *selects uniformly random bit* $\mathsf{b} \xleftarrow{\$} \{0,1\}$*, then sends challenge ciphertext* $\mathsf{ct_b} = \mathsf{Enc}(\mathsf{m_b}, \mathsf{pk})$ *to* $\mathcal{A}$
- *The adversary* $\mathcal{A}$ *outputs a guess bit* $\mathsf{b}' \in \{0,1\}$

Definition 6. *(One-more unforgeability (OMU) [108]) It is a special type of unforgeability notion for blind signatures. OMU ensures that an adversary cannot forge at*

least one valid signature for a message of its choosing. Formally, it means any efficient adversary's advantage in the following game is negligible:

- *The challenger* $\mathcal{C}$ *generates the public/secret key pair (*pk, sk*), then sends public key* pk *to adversary* $\mathcal{A}$*.*
- *The adversary* $\mathcal{A}$ *queries* l*-many times the signing oracle* $\mathcal{O}_{\mathsf{Sign}}(pk)$ *on input* $\mathsf{m_i}$ *(*$1 \leq \mathsf{i} \leq \mathsf{l}$*), then takes signatures on them as* $\{\sigma(\mathsf{m_i})\}_{\mathsf{i}=1}^{\mathsf{l}}$
- *Finally,* $\mathcal{A}$ *outputs* $(\mathsf{l}+1)$*-many message-signature pair as* $\{(\mathsf{m_i}, \sigma(\mathsf{m_i}))\}_{\mathsf{i}=1}^{\mathsf{l}+1}$

If $\mathsf{Verify}(\{\mathsf{m_i}, \sigma(\mathsf{m_i})\}_{\mathsf{i}=1}^{\mathsf{l}+1}) = 1$ *and all* $\mathsf{m_i}$ *are distinct, then adversary breaks one-more unforgeability of the signature.*

The next subsections give the specific cryptographic building blocks employed in Proba. To begin, since Proba is a blockchain-based voting system, digital signatures are required for all transactions. Second, since Proba is also built on the Helios system, it employs homomorphic encryption and zero-knowledge proofs. Third, Proba leverages the threshold-issuance anonymous credential system to address privacy and robustness problems. Finally, Proba manages transaction fees via a consortium blockchain. In general, λ represents the chosen scheme's security parameter, whereas $\mathsf{negl}(\lambda)$ denotes a function with insignificant value.

3.1.1 Digital Signatures

Digital signatures prove the message's origin, thereby providing authentication for schemes. They depend on the message and the signer; thus, later modifications in the message or attaching the same signature to any message become meaningless.

Definition 7 (Digital Signature schemes)**.** *For a given global public parameter* pp*, a digital signature (DS) scheme contains three probabilistic polynomial time (PPT) algorithms as follows:*

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp})$*: This key generation algorithm takes the global public parameters* pp *and returns the pair of signing (secret) - verification (public) keys* $(\mathsf{sk}, \mathsf{vk})$ *associated with a message space* $\mathcal{M}$*.*

- $(\sigma) \leftarrow \mathcal{DS}.\mathsf{Sign}(\mathsf{sk}, m)$*: On input of the signing key* sk *and a message* $m \in \mathcal{M}$*, Sign algorithm outputs a signature* σ*.*

- $(0,1) \leftarrow \mathcal{DS}.\mathsf{Verify}(\mathsf{vk}, \sigma, m)$*: This deterministic algorithm takes as inputs the verification key* vk*, a signature* σ *and* m *and outputs either* 1 *(accept) or* 0 *(reject).*

The primary security requirements for a signature scheme are *correctness* and *unforgeability against chosen message attack (EUF-CMA)*, which are defined as follows:

Definition 8 (Correctness)**.** *A digital signature is called correct if we have:*

$$\Pr\begin{bmatrix}\forall\ (\mathsf{sk},\mathsf{vk}) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp}), m \in \mathcal{M}: \\ \mathcal{DS}.\mathsf{Verify}\left(\mathsf{vk}, m, \mathcal{DS}.\mathsf{Sign}(\mathsf{sk}, m)\right) = 1\end{bmatrix} \geq 1 - \mathsf{negl}(\lambda)$$

Definition 9 ($\mathsf{EUF}-\mathsf{CMA}$)**.** *A signature scheme is* $\mathsf{EUF}-\mathsf{CMA}$ *secure if for all PPT adversaries* $\mathcal{A}$ *with some oracle we have the following advantage,* $Adv_{\mathcal{DS}}^{\mathsf{EUF}-\mathsf{CMA}}(\mathcal{A})$*:*

$$\Pr\begin{bmatrix}\forall\ (\mathsf{sgk},\mathsf{vk}) \leftarrow \mathcal{DS}.\mathsf{KGen}(pp), \\ (\sigma^*, m^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(pp, vk): \\ m^* \notin \mathcal{Q}_{\mathsf{S}}\ \wedge\ \mathsf{Verify}(\mathsf{vk}, \sigma^*, m^*) = 1\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

The signature oracle $\mathcal{O}_{\mathsf{Sign}}$ *takes a message* $m \in \mathcal{M}$*, and runs* $\mathsf{Sign}(\mathsf{sgk}, m)$ *then adds the message to a query set* $\mathcal{Q}_{\mathsf{S}}$*.*

3.1.2 Homomorphic Encryptions

Generally, Homomorphic Encryption (HE) is an encryption algorithm capable of performing certain computations over ciphertexts. The Partial HE schemes allow only an unlimited number of operations of only one type: either addition or multiplication.

Definition 10 (Homomorphic Encryptions)**.** *HE scheme contains the following PPT algorithms:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{HE}.\mathsf{KGen}(\mathsf{pp})$*: Key generation is a randomized algorithm that*

takes public parameter pp *as input and returns a pair of public/secret keys* $(\mathsf{pk}, \mathsf{sk})$ *as outputs.*

- $(\mathsf{ct}) \leftarrow \mathcal{HE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m)$*: The encryption algorithm is a randomized algorithm which takes the public parameters* pp*, public key* pk*, and a message* $m \in \mathcal{M}$ *as inputs. It outputs a ciphertext* $\mathsf{ct} \in \mathcal{C}$*, ciphertext space.*

- $(\perp, m) \leftarrow \mathcal{HE}.\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$*: The decryption algorithm takes the public parameters* pp*, secret key* sk*, and ciphertext* ct*, then returns message* m *as a valid output, otherwise* $\perp$

The primary security requirements for a HE scheme are *correctness* and *indistinguishability against chosen plaintext attack (IND-CPA)*, which are defined as follows:

Definition 11 (Correctness)**.** *A homomorphic encryption scheme is called correct if we have:*

$$\Pr\begin{bmatrix}\forall\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{HE}.\mathsf{KGen}(\mathsf{pp}), m \in \mathcal{M}: \\ \mathcal{HE}.\mathsf{Dec}\left(\mathsf{pp}, \mathsf{sk}, \mathcal{HE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m)\right) = m\end{bmatrix} \geq 1 - \mathsf{negl}(\lambda)$$

Definition 12 ($\mathsf{IND}-\mathsf{CPA}$)**.** *A homomorphic encryption scheme is* $\mathsf{IND}-\mathsf{CPA}$ *secure if for all PPT adversaries* $\mathcal{A}$ *with an oracle we have the following advantage,* $Adv^{\mathsf{IND}-\mathsf{CPA}}_{\mathcal{HE}}(\mathcal{A})$*:*

$$\Pr\begin{bmatrix}\forall\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{HE}.\mathsf{KGen}(pp), \\ \mathsf{ct}^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}}(m^*, pk) : m^* \notin \mathcal{Q}_{\mathcal{HE}} \wedge \\ \mathcal{HE}.\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*) \neq \perp\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

The homomorphic encryption oracle $\mathcal{O}_{Enc}$ *takes a message* $m \in \mathcal{M}$*, runs* $\mathcal{HE}.\mathsf{Enc}$ (pk, pp, m) *and adds the message to a query set* $\mathcal{Q}_{\mathcal{HE}}$*.*

3.1.3 Zero Knowledge Proofs

Zero Knowledge Proofs (ZKPs) are useful two-party protocols between the prover and the verifier. The prover proves the knowledge of the statement to the verifier

without disclosing anything more than true or false. The interaction between the verifier and the prover in ZKPs can be eliminated with non-interactive zero-knowledge proofs like the well-known Fiat-Shamir transformation [48].

Definition 13 (Zero Knowledge Proofs)**.** *ZKP comprises the following two algorithms:*

- $\pi \leftarrow \mathcal{ZKP}.Pr(x)$*: Prove algorithm takes as input statement* x*, then output a proof* π *that consists of witness commitment* com*, challenge* c*, and response* $resp$*.*
- $(0,1) \leftarrow \mathcal{ZKP}.Vr(x,\pi)$*: Verify algorithm takes a proof* π *and statement* x*, then output* 1 *as an accept status, otherwise* 0*.*

Three properties of ZKP, that is *completeness*, *soundness* and *zero-knowledge*, are defined as follows:

Definition 14 (Completeness)**.** *Completeness ensures that if two parties follow the protocol, the verifier accepts the proof. A ZKP scheme satisfies completeness if we have the following:*

$$\Pr\begin{bmatrix}\pi \leftarrow \mathcal{ZKP}.Pr(x), \\ \mathcal{ZKP}.Vr\,(x,\pi)=1\end{bmatrix} \geq 1-\mathsf{negl}(\lambda)$$

Definition 15 (Soundness)**.** *Soundness prevents the verifier from accepting false proof of the statement. A ZKP scheme satisfies soundness if for all PPT adversaries* $\mathcal{A}$ *we have the following advantage,* $Adv^{sound}_{\mathsf{ZKP}}(\mathcal{A})$*:*

$$\Pr\begin{bmatrix}\pi^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{ZKP}}}(x^*), x^* \notin \mathcal{Q}_{\mathsf{ZKP}}, \\ \mathsf{ZKP}.Vr\,(x^*,\pi^*)=1\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

The ZKP oracle $\mathcal{O}_{\mathcal{ZKP}}$ takes a statement x and witness w, runs $\mathcal{ZKP}.\mathsf{Pr}$ and adds the statement x to a query set $\mathcal{Q}_{\mathcal{ZKP}}$.

Definition 16 (Zero-knowledge)**.** *The zero-knowledge property states that proof does not leak any information except for the truth of the statement. A ZKP scheme satisfies*

zero-knowledge if we have:

$$\Pr\begin{bmatrix}\pi \leftarrow \mathsf{ZKP}.Pr(x)\\ \pi^* \leftarrow \mathcal{S}^{\mathcal{O}_{Sim}}(x^*)\\ \mathcal{ZKP}.Vr\,(x,\pi)=1\\ \mathcal{ZKP}.Vr\,(x^*,\pi^*)=1\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

In this property, Simulator $\mathcal{S}^{\mathcal{O}_{\mathsf{Sim}}}$ algorithm is used to generate a proof that is indistinguishable from the real proof without using prover's secret information. Thus, no extra information is obtained.

3.1.4 Threshold-Issuance Anonymous Credentials

Anonymous credentials make the authentication process possible while preserving the user's privacy. Thus, they are useful for many privacy-preserving applications. However, since they rely on a single issuer to generate the credentials, there is a risk of a single point of failure. To eliminate this weakness, Sonnino et al. proposed Coconut [129] called Threshold-Issuance Anonymous Credentials (TIAC). TIAC expands the Pointcheval and Sanders signatures [107] in a threshold form by taking advantage of hash functions, thus making it possible for some set of credential issuers to issue credentials jointly.

TIAC is an optional declaration credential construction supporting distributed threshold issuance. Unlinkable optional attribute disclosures and public and private attributes are supported by this protocol even when a part of issuing authorities is malicious or offline. Recently, Rial et al. [113] have analyzed the security properties of TIAC by introducing an ideal functionality that captures all the security properties of a threshold blind signature. They introduced a new construction that follows TIAC with a few modifications to realize ideal functionality. They have some changes for issuing blind signatures and for signature shows.

In $\mathcal{TIAC}$ public signature parameter is taken as $\mathsf{params} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \mathsf{p}, \mathsf{e}, \mathsf{g}_1, \mathsf{h}_1, \mathsf{g}_2)$, where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ refers to bilinear map with order p, $\mathsf{g}_1, \mathsf{h}_1$ be generators of $\mathbb{G}_1$,

and g_2 be generators of $\mathbb{G}_2$. The improved $\mathcal{TIAC}$ scheme [113] encompasses the following probabilistic polynomial-time (PPT) functions:

- $\mathcal{TIAC}.\mathsf{KGen}(\mathsf{params}) \rightarrow (\mathsf{msgk}, \mathsf{mvk})$: In a simple key generation function with a Third Trusted Party (TTP), TTP takes params, picks two polynomial f_1, f_2 of degree $t-1$ then set the master signing key as $\mathsf{msgk} = (x, y) = (f_1(0), f_2(0))$ and master verification key as $\mathsf{mvk} = (g_2{}^x, g_2{}^y, g_1{}^y) = (\alpha_2, \beta_2, \beta_1)$. Then, TTP issues to each authority a signing key $\mathsf{sgk}_i = (x_i, y_i) = (f_1(i), f_2(i))$ and publish its verification key $\mathsf{vk}_i = (g_2^{x_i}, g_2^{y_i}, g_1^{y_i}) = (\alpha_{2,i}, \beta_{2,i}, \beta_{1,i})$. The key generation function can be also executed without TTP as in [54].

- $\mathcal{TIAC}.\mathsf{IsCred}(m, \phi) \rightarrow \sigma_i(m)$: The issue credential function consists of three sub-functions:

  - $\mathcal{TIAC}.\mathsf{PrepBSign}(m, \phi, r_v) \rightarrow (m', \phi)$: Prepare blind signature function converts the private attribute m to blinded form m' using a blind factor r. Firstly, the user generates the Pedersen commitment [101] $com_v = g_1^{o_v} \cdot h_1^m$ where o_v is the randomly chosen value, and set $\mathsf{Hash}(com_v) = h_v$ to build common basis. Then the user produces $com = g_1^{r_v} \cdot h_v^m$ on the same m where r_v is random blind factor. Later, the user proceeds to generate the ZKP on it labeled as π_v using knowledge of representation check as in [145]. This ZKP proves that the committed values are well-formed and m committed in com_v is the same as the message committed in com. Thus, algorithm outputs $m' = (h_v, com_v, com, \pi_v)$ and the application-specific statement ϕ.
  - $\mathcal{TIAC}.\mathsf{BSign}(\mathsf{sgk}_i, m', \phi) \rightarrow \sigma_i(m')$: This function blindly sign the output of $\mathcal{TIAC}.\mathsf{PrepBSign}$ with signing key sgk_i, and outputs partial blind signature $\sigma_i(m')$. In detail, each authority checks $\mathsf{Hash}(com_v)$ and the proof π_v, then builds partial signature as $\sigma_i(m') = (h_v, c_i = (h_v^{x_i} \cdot com^{y_i}))$.
  - $\mathcal{TIAC}.\mathsf{UnBlind}(\sigma_i(m'), r_v) \rightarrow \sigma_i(m)$: UnBlind function is used to remove blinding factor r_v and get an original partial signature/credential $\sigma_i(m)$. To remove blinding factor, the user computes $(h_v, c_i \cdot \beta_{1,i}^{-r_v}) = (h_v, s_i)$ as $\sigma_i(m)$.

- $\mathcal{TIAC}.\mathsf{AggCred}(\sigma_1(m), \ldots, \sigma_t(m)) \rightarrow \sigma(m)$: Any subset of t partial creden-

tial aggregated into single credential $\sigma(\mathsf{m})$. In this function, the user first verifies the validity of the acquired signature $\sigma_{\mathsf{i}}(\mathsf{m})$ by checking bilinear pairing $\mathsf{e}(\mathsf{h_v}, \alpha_{2,\mathsf{i}} \cdot \beta_{2,\mathsf{i}}^{\mathsf{m}}) = \mathsf{e}(\mathsf{s_i}, \mathsf{g_2})$, then computes $\sigma(\mathsf{m}) = (\mathsf{h_v}, \prod_{\mathsf{i}=1}^{\mathsf{t}} \mathsf{s_i}^{\mathsf{l_i}}) = (\mathsf{h_v}, \mathsf{s_v})$ where $\mathsf{l_i}$ is the Lagrange coefficient, $\mathsf{l_i} = (\prod_{\mathsf{j}=1,\mathsf{j}\neq\mathsf{i}}^{\mathsf{t}}(0-\mathsf{j}))(\prod_{\mathsf{j}=1,\mathsf{j}\neq\mathsf{i}}^{\mathsf{t}}(\mathsf{i}-\mathsf{j}))^{-1}$ mod p.

- $\mathcal{TIAC}.\mathsf{PrCred}(\sigma(\mathsf{m}), \mathsf{mvk}, \mathsf{m}, \phi') \rightarrow (\sigma''(\mathsf{m}), \phi')$: The prove credential function takes $\sigma(\mathsf{m})$, mvk, m, and application-specific new statement ϕ', then outputs randomized signature $\sigma''(\mathsf{m})$ and the statement ϕ' that shows some proofs of possession of the credential. In the detail, the user selects two random elements $\mathsf{r}', \mathsf{r}''$ and randomize the signature as $\sigma'' = (\mathsf{h_v}^{\mathsf{r}'}, \mathsf{s_v}^{\mathsf{r}'} \cdot (\mathsf{h_v}'')^{\mathsf{r}''}) = (\mathsf{h_v}'', \mathsf{s_v}'')$. Then the user also generates the value $\mathsf{k} = \alpha_2 \cdot \beta_2^{\mathsf{m}} \cdot \mathsf{g_2}^{\mathsf{r}''}$ and the proof π_{v}' to prove k is well-formed and the message fulfills the application-specific statement ϕ'. At the end, randomized signature $\sigma''(\mathsf{m})$ consists of the tuple $(\sigma'', \mathsf{k}, \pi_{\mathsf{v}}')$.

- $\mathcal{TIAC}.\mathsf{VrCred}(\mathsf{mvk}, \sigma''(\mathsf{m}), \phi') \rightarrow (0, 1)$: The credential verification function takes mvk, a randomized credential $\sigma''(\mathsf{m})$, application-specific statement ϕ', and outputs 1 as an verified credential, otherwise 0. In this function, the verifier checks the randomized signature by $\mathsf{e}(\mathsf{h_v}'', \mathsf{k}) = \mathsf{e}(\mathsf{s_v}'', \mathsf{g_2})$, and checks the proof π_{v}' using mvk and ϕ'. Based on these verifications, the verifier outputs 1 or 0.

The TIAC scheme ensures three important security requirements including *unforgeability*, *unlinkability*, and *blindness* [129]. While unforgeability means the adversarial user's inability to convince an honest verifier of the possession of the fake credential, blindness prevents an adversarial authority from learning any information on attribute m during BSign phase of IsCred. Also, unlinkability means the adversarial verifier's inability to link the execution of PrCred with another execution of PrCred, or with the execution of IsCred. These requirements can be defined as follows:

Definition 17 (Unforgeability)**.** *A TIAC scheme satisfies unforgeability (as long as*

less than threshold t authorities collude) if we have negligible probability:

$$\Pr\left[\begin{array}{l}(\mathsf{msgk},\mathsf{mvk}) \leftarrow \mathcal{TIAC}.\mathsf{KGen}(\mathsf{params}),\\ \sigma(\mathsf{m}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{IsCred}}}(\mathsf{m},\phi) \textit{ for some } m \in \mathcal{Q}_{\mathsf{Cred}}\\ \textit{Output a new pair } (\mathsf{m}^*,\sigma(\mathsf{m}^*)),\\ (\sigma''(\mathsf{m}^*),\phi^*) \leftarrow \mathcal{TIAC}.\mathsf{PrCred}(\sigma(\mathsf{m}^*),\mathsf{mvk},\mathsf{m}^*,\phi^*):\\ \mathcal{TIAC}.\mathsf{VrCred}\left(\mathsf{mvk},\sigma''(\mathsf{m}^*),\phi^*\right)=1,\\ m^* \notin \mathcal{Q}_{\mathsf{Cred}}\end{array}\right]$$

$\mathcal{A}^{\mathcal{O}_{\mathsf{IsCred}}}$ is an oracle that executes $\mathcal{TIAC}.\mathsf{IssueCred}$ protocol. Also, a query set $\mathcal{Q}_{\mathsf{Cred}}$ represents valid private attributes m. According to this definition:

- The challenger $\mathcal{C}$ runs key generation algorithm, then sends master verification key mvk to adversary $\mathcal{A}$.
- $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathsf{IsCred}}$ with different m and ϕ, then gets signature on them as $\sigma(\mathsf{m})$.
- Finally, $\mathcal{A}$ outputs a candidate forgery message/signature pair $(\mathsf{m}^*,\sigma(\mathsf{m}^*))$, and generates the randomized signature with related information $(\sigma''(\mathsf{m}^*),\phi^*)$ on it.

If $\mathcal{TIAC}.\mathsf{VrCred}(\mathsf{mvk},\sigma''(\mathsf{m}^*),\phi^*)=1$, and m^* is a new message that is not queried before, then adversary forges the signature.

Definition 18 (Blindness)**.** *A TIAC scheme satisfies blindness if we have the probability that is negligibly close to $\frac{1}{2}$ where*

$$\Pr\left[\begin{array}{l}\textit{For a given } \mathsf{params},\\ (\mathsf{m}_0^{'},\mathsf{m}_1^{'}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{choose}}}(\mathsf{params}),\\ \sigma_{\mathsf{i}}(\mathsf{m}_0^{'}) \leftarrow \mathcal{TIAC}.\mathsf{BSign}(\mathsf{sgk}_{\mathsf{i}},\mathsf{m}_0^{'},\phi),\\ \sigma_{\mathsf{i}}(\mathsf{m}_1^{'}) \leftarrow \mathcal{TIAC}.\mathsf{BSign}(\mathsf{sgk}_{\mathsf{i}},\mathsf{m}_1^{'},\phi),\\ \textit{Given } (\sigma_{\mathsf{i}}(\mathsf{m}_{\mathsf{r}}^{'}),\sigma_{\mathsf{i}}(\mathsf{m}_{1-\mathsf{r}}^{'})),\mathsf{r}\in\{0,1\}:\\ \mathsf{r}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{guess}}}(\sigma_{\mathsf{i}}(\mathsf{m}_{\mathsf{r}}^{'}),\sigma_{\mathsf{i}}(\mathsf{m}_{1-\mathsf{r}}^{'})),\mathsf{r}'=\mathsf{r}\end{array}\right]$$

According to this definition:

- For a given params, Adversary $\mathcal{A}$ chooses two blind messages $\mathsf{m}_0^{'}$ and $\mathsf{m}_1^{'}$ with the same length, then sends them to challenger $\mathcal{C}$
- The challenger $\mathcal{C}$ signs the two messages blindly as $\sigma(\mathsf{m}_0^{'})$ and $\sigma(\mathsf{m}_1^{'})$, then send them in uncertain order.

If $\mathcal{A}$ guesses who signed the first, then he can distinguish the order of these two signatures, thus $\mathcal{A}$ breaks blindness.

Definition 19 (Unlinkability)**.** *A TIAC scheme satisfies unlinkability if we have a probability that is negligibly close to* $\frac{1}{2}$ *where:*

$$\Pr\left[\begin{array}{l} \textit{For a given } \mathsf{params}, \\ \mathsf{m} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{choose}}}(\mathsf{params}), \\ \textit{For } b \xleftarrow{\$} \{0,1\}, \\ \sigma_0 = \sigma(\mathsf{m}) \leftarrow \mathcal{TIAC}.\mathsf{IsCred}(\mathsf{m}, \phi), \\ \sigma_1 = (\sigma''(\mathsf{m}), \phi_1') \leftarrow \mathcal{TIAC}.\mathsf{PrCred}(\sigma(\mathsf{m}), \mathsf{mvk}, \mathsf{m}, \phi_1') : \\ \mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{guess}}}(\sigma_{\mathsf{b}}'') \wedge \mathsf{b}' = \mathsf{b} \end{array}\right]$$

According to this definition:

- For a given params, $\mathcal{A}$ chooses a message m, then sends it to challenger.
- $\mathcal{C}$ computes the signature as follows:
  - if $\mathsf{b} = 0$, σ_0 will be $\sigma(\mathsf{m}) \leftarrow \mathcal{TIAC}.\mathsf{IsCred}(\mathsf{m}, \phi)$.
  - if $\mathsf{b} = 1$, σ_1 will be $(\sigma''(m), \phi_1') \leftarrow \mathcal{TIAC}.\mathsf{PrCred}(\sigma(\mathsf{m}), \mathsf{mvk}, \mathsf{m}, \phi_1')$

  then sends it to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess bit $\mathsf{b}' \in \{0,1\}$.

If $\mathcal{A}$ guesses correctly two signatures, one randomized by PrCred algorithm and the other normal by IsCred algorithm, then $\mathcal{A}$ breaks the unlinkability of the signatures.

3.1.5 Consortium Blockchain

In Blockchain-powered Helios, Ethereum public blockchain is used as a platform. It has no centralized management, and anyone can join the network. However, transaction duration in the public blockchain is long, and some transaction fees are required for successful transactions. These flaws have a substantial impact on the quality of the elections. To enhance scalability and reduce transaction fees, it is imperative to replace public blockchains with private or consortium blockchains. These blockchains exclusively permit authorized individuals to join their networks. Nevertheless, the main difference is that a single authority manages private blockchain, whereas multiple authorities manage consortium blockchain. Since the centralization in the private blockchain is an undesirable property in the election systems, the proposed system infrastructure is based on a consortium blockchain. Thus, traffic in the network and costs for the transaction are solved by a semi-decentralized blockchain type.

3.2 The Proposed System Architecture

3.2.1 Entities

The entities of Proba are as follows:

- **Election Administrators (EAs)** authenticate the voters and authorize the voter's public wallet keys, thereby, their wallet addresses. Also, they generate the election key in a distributed manner and are in charge of decrypting the aggregated votes partially and computing the final tally function.
- **Voters** are responsible for generating their wallet key/address to participate in the election, where each candidate is represented by their unique number.
- **Blockchain** serves as PBB in a decentralized way and checks the well-formedness of the received ballots prior to casting them.

For the systems, the attacker/threat model needs to be defined to specify the attacker's capabilities. There are various attacker models for different election levels [60].

These levels are determined by political importance [74]. Since Proba is not designed for a large-scale political election, we assume the attacker is a basic honest but curious model.

3.2.2 Proba System Construction

The general architecture of Proba in Figure 3.1 can be described as follows:

1. Each voter needs to generate their wallet address without interacting with EA.
2. After the generation of the wallet, voters send their real ID in plain and their wallet's public key in blind form to EA.
3. Receiving these values, EA checks the voter's real ID for authentication, then signs the voter's blind public key for authorization to be used as a credential in the election.
4. When the election begins, voters initiate a transaction that contains their encrypted votes and related proofs, as well as their randomized credentials.
5. Once the transaction is received, blockchain sends the transaction ID to voter.
6. After each voter takes the transaction ID and the election closes, EA comes to decrypt the votes partially. In addition to partially decrypted votes, the proof of correct decryption is published by EA.
7. Upon receiving proofs and partial decryptions, the smart contract checks the proofs, then uses homomorphic encryption property to get the tallying result. Finally, the result is published by the smart contract.

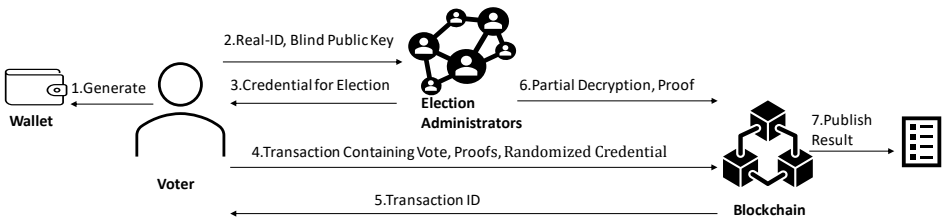

Figure 3.1: General architecture of Proba

In Proba, voters have the right to re-vote through their addresses, and the last vote will be counted. However, a timeout requirement on the public key is set to prevent

excessive transaction requests. After sending the transaction, the voter needs to wait enough time to re-vote again.

The sequence diagram of Proba is provided in Figure 3.2 with comprehensive details. The diagram shows the pre-election process through the steps marked with number 1, while the subsequent steps marked with number 2 represent the election period. In addition, the steps starting with number 3 show the processes after the election.

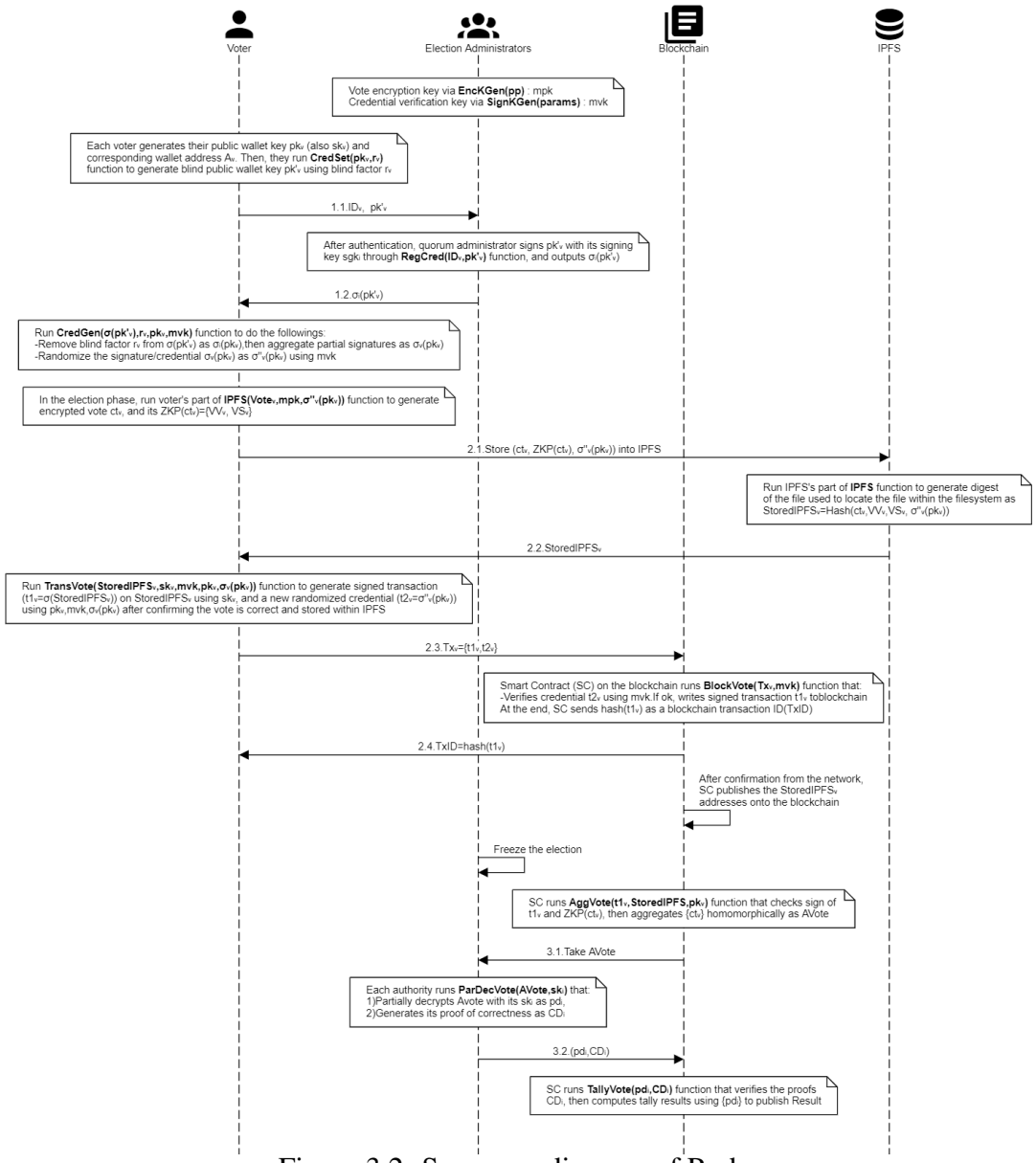

Figure 3.2: Sequence diagram of Proba

Also, the formal construction of the used functionalities in Algorithm 1 can be classified as pre-election, election, and post-election. It is worth noting that to simplify

the notation, we will not include the application-specific statement ϕ and the proof π_{v} specified in $\mathcal{TIAC}$ for the generic construction of Proba in Algorithm 1.

Algorithm 1: Proba: Generic Construction

Proba: Generic Construction

Function EncKGen (pp) **:**
- $\mathsf{mpk} \leftarrow 1$
- **for** $\mathsf{i} \leftarrow 1$ **to** $\mathsf{n_e}$ **do**
  - $\mathcal{HE}.\mathsf{KGen}(\mathsf{pp}) \rightarrow (\mathsf{pk_i}, \mathsf{sk_i})$
  - $\mathsf{PoK_i} \leftarrow \mathcal{ZKP}.\mathsf{Pr}(\mathsf{sk_i})$
  - **if** $\mathcal{ZKP}.\mathsf{Vr}(\mathsf{PoK_i}) = 1$ **then**
    - $\mathsf{mpk} \leftarrow \mathsf{mpk} \cdot \mathsf{pk_i}$
  - **return** mpk

Function SignKGen (params) **:**
- **for** $\mathsf{i} \leftarrow 1$ **to** $\mathsf{n_e}$ **do**
  - $\mathcal{TIAC}.\mathsf{KGen}(\mathsf{params}) \rightarrow (\mathsf{msgk}, \mathsf{mvk})$
- **return** mvk

Function CredSet ($\mathsf{pk_v}, \mathsf{r_v}$) **:**
- $\mathcal{TIAC}.\mathsf{PrepBSign}(\mathsf{pk_v}, \mathsf{r_v}) \rightarrow \mathsf{pk'_v}$
- **return** $\mathsf{pk'_v}$

Function RegCred ($\mathsf{ID_v}, \mathsf{pk'_v}$) **:**
- **if** $\mathsf{ID_v}$ *is valid* **then**
  - **for** $\mathsf{i} \leftarrow 1$ **to** t **do**
    - $\mathcal{TIAC}.\mathsf{BSign}(\mathsf{pk'_v}, \mathsf{sgk_i}) \rightarrow \sigma_{\mathsf{i}}(\mathsf{pk'_v})$
  - **return** $\sigma_{\mathsf{v}}(\mathsf{pk'_v}) = \{\sigma_{\mathsf{i}}(\mathsf{pk'_v})\}_{i=1}^{t}$

Function CredGen ($\sigma_{\mathsf{v}}(\mathsf{pk'_v}), \mathsf{r_v}, \mathsf{pk_v}, \mathsf{mvk}$) **:**
- **for** $\mathsf{i} \leftarrow 1$ **to** t **do**
  - $\mathcal{TIAC}.\mathsf{UnBSign}(\sigma_{\mathsf{i}}(\mathsf{pk'_v}), \mathsf{r_v}) \rightarrow \sigma_{\mathsf{i}}(\mathsf{pk_v})$
- $\mathcal{TIAC}.\mathsf{AggCred}(\{\sigma_{\mathsf{i}}(\mathsf{pk_v})\}_{i=1}^{t}) \rightarrow \sigma_{\mathsf{v}}(\mathsf{pk_v})$
- $\mathcal{TIAC}.\mathsf{PrCred}(\sigma_{\mathsf{v}}(\mathsf{pk_v}), \mathsf{mvk}, \mathsf{pk_v}) \rightarrow \sigma''_{\mathsf{v}}(\mathsf{pk_v})$
- **return** $\sigma''_{\mathsf{v}}(\mathsf{pk_v})$

Function IPFS ($\mathsf{Vote_v}, \mathsf{mpk}, \sigma''_{\mathsf{v}}(\mathsf{pk_v})$) **:**
- $\mathcal{HE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{Vote_v}) \rightarrow \mathsf{ct_v}$
- $\mathcal{ZKP}.\mathsf{PrVV}(\mathsf{ct_v}) \rightarrow \mathsf{VV_v}$
- $\mathcal{ZKP}.\mathsf{PrVS}(\mathsf{ct_v}) \rightarrow \mathsf{VS_v}$
- $\mathsf{StoredIPFS_v} \leftarrow \mathsf{Hash}(\mathsf{ct_v}, \mathsf{VV_v}, \mathsf{VS_v}, \sigma''_{\mathsf{v}}(\mathsf{pk_v}))$
- **return** $\mathsf{StoredIPFS_v}$

Function TransVote ($\mathsf{StoredIPFS_v}, \mathsf{sk_v}, \mathsf{pk_v}, \sigma_{\mathsf{v}}(\mathsf{pk_v})$)
- $\mathsf{StoredIPFS^*_v} \leftarrow \mathsf{IPFS}(\mathsf{Vote_v}, \mathsf{mpk}, \sigma''_{\mathsf{v}}(\mathsf{pk_v}))$
- **if** $\mathsf{StoredIPFS^*_v} = \mathsf{StoredIPFS_v}$ **then**
  - $\mathsf{t1_v} \leftarrow \mathcal{DS}.\mathsf{Sign}(\mathsf{sk_v}, \mathsf{StoredIPFS_v})$
  - $\mathsf{t2_v} \leftarrow \mathcal{TIAC}.\mathsf{PrCred}(\sigma_{\mathsf{v}}(\mathsf{pk_v}), \mathsf{mvk}, \mathsf{pk_v}$
- **return** $\mathsf{Tx_v} = (\mathsf{t1_v}, \mathsf{t2_v})$

Function BlockVote ($\mathsf{Tx_v}, \mathsf{mvk}$) **:**
- **if** $\mathcal{TIAC}.\mathsf{VrCred}(\mathsf{mvk}, \mathsf{t2_v}) = 1$ **then**
  - Write transaction to the blockchain
- **return** $\mathsf{Tx_{ID}} = \mathsf{Hash}(\mathsf{t1_v})$

Function AggVote ($\mathsf{t1_v}, \mathsf{StoredIPFS}, \mathsf{pk_v}$) **:**
- **if** $\mathcal{DS}.\mathsf{Vr}(\mathsf{pk_v}, \mathsf{t1_v}, \mathsf{StoredIPFS}) = 1$ **then**
  - **if** $\mathcal{ZKP}.\mathsf{Vr}(\mathsf{VV_v}) = 1 \wedge \mathcal{ZKP}.\mathsf{Vr}(\mathsf{VS_v}) = 1$ **then**
    - $\mathsf{AVote} \leftarrow 1$
    - **for** $\mathsf{v} \leftarrow 1$ **to** $\mathsf{n_v}$ **do**
      - $\mathsf{AVote} = \mathsf{AVote} \cdot \mathsf{ct_v}$
    - **return** AVote

Function ParDecVote ($\mathsf{AVote}, \mathsf{pp}, \mathsf{sk_i}$) **:**
- **for** $\mathsf{i} \leftarrow 1$ **to** $\mathsf{n_e}$ **do**
  - $\mathsf{pd_i} \leftarrow \mathcal{HE}.\mathsf{ParDec}(\mathsf{sk_i}, \mathsf{AVote})$
  - $\mathsf{CD_i} \leftarrow \mathcal{ZKP}.Pr(\mathsf{pd_i})$
- **return** $(\mathsf{pd_i}, \mathsf{CD_i})$

Function TallyVote ($\mathsf{pd_i}, \mathsf{CD_i}$) **:**
- **for** $\mathsf{i} \leftarrow 1$ **to** t **do**
  - **if** $\mathcal{ZKP}.\mathsf{Vr}(\mathsf{CD_i}) = 1$ **then**
    - $\mathsf{Result} \leftarrow \mathcal{HE}.\mathsf{Dec}(\mathsf{pd_i}, \mathsf{AVote})$
  - **return** Result

3.2.2.1 Pre-election

In the pre-election stage, the functions $\mathsf{EncKGen}$ and $\mathsf{SignKGen}$ are responsible for producing the master public key for the encryption scheme and the master verification key for threshold anonymous credentials, respectively. Additionally, $\mathsf{CredSet}$ is responsible for generating the blind form of the voter's wallet public key, while $\mathsf{RegCred}$ signs this key blindly. Subsequently, the voter obtains the original signature on the wallet public key to employ it as a credential through the use of the $\mathsf{CredGen}$ function.

- $\mathsf{EncKGen}(\mathsf{pp}) \rightarrow \mathsf{mpk}$: It is distributed function run by $\{\mathsf{EA_i}\}_{i=1}^{n_e}$ for the generation of master public (encryption) key mpk on global public encryption parameter (pp) input. As in distributed key generation [54], each $\mathsf{EA_i}$ generates its public key $\mathsf{pk_i}$ and secret key $\mathsf{sk_i}$ (via $\mathcal{HE}.\mathsf{KGen}$), then publish $\mathsf{pk_i}$ with proof of knowledge of secret key $\mathsf{PoK_i}$. Once the proofs are verified, mpk is generated by multiplying all published valid public keys, and this value will be used to encrypt the vote.

- $\mathsf{SignKGen}(\mathsf{params}) \rightarrow \mathsf{mvk}$: In a simple key generation function, TTP generates the master signing key msgk and master verification keys mvk, as well as the shares of EA's signing keys $\mathsf{sgk_i}$ with $\mathcal{TIAC}.\mathsf{KGen}$. Nonetheless, it is worth noting that this function can also be executed without the involvement of the TTP, as previously mentioned.

- $\mathsf{CredSet}(\mathsf{pk_v}, \mathsf{r_v}) \rightarrow \mathsf{pk'_v}$: Credential setup function is run by the voter $\{\mathsf{V_v}\}_{v=1}^{n_v}$. Taking wallet's public key $\mathsf{pk_v}$ and random blinding factor $\mathsf{r_v}$, voter generates blind wallet public key $\mathsf{pk'_v}$ using $\mathcal{TIAC}.\mathsf{PrepBSign}$.

- $\mathsf{RegCred}(\mathsf{ID_v}, \mathsf{pk'_v}) \rightarrow \sigma_\mathsf{v}(\mathsf{pk'_v})$: Registration credential function is run by any quorum subset t of EA. A quorum of EA checks the voter's ID information $\mathsf{ID_v}$ and related proofs π_v. If ok, EA signs blind wallet public key $\mathsf{pk'_v}$ partially as $\sigma_\mathsf{i}(\mathsf{pk'_v})$ using $\mathcal{TIAC}.\mathsf{BSign}$, then outpus $\{\sigma_\mathsf{i}(\mathsf{pk'_v})\}_{i=1}^{t} = \sigma_\mathsf{v}(\mathsf{pk'_v})$.

- $\mathsf{CredGen}(\sigma_\mathsf{v}(\mathsf{pk'_v}), \mathsf{r_v}, \mathsf{pk_v}, \mathsf{mvk}) \rightarrow \sigma''_\mathsf{v}(\mathsf{pk_v})$: Credential generation function is run by the voter $\{\mathsf{V_v}\}_{v=1}^{n_v}$. Taking t partial signature on blind wallet public key

as $\sigma_{\mathsf{v}}(\mathsf{pk}'_{\mathsf{v}}) = \{\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}})\}_{\mathsf{i}=1}^{\mathsf{t}}$, each voter removes the blinding factor r_{v} and get the original partial signatures $\{\sigma_{\mathsf{i}}(\mathsf{pk}_{\mathsf{v}})\}_{\mathsf{i}=1}^{\mathsf{t}}$ using $\mathcal{TIAC}$.UnBSign. Then, the voter uses these partial quorum signatures to generate the main aggregated signature $\sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ via $\mathcal{TIAC}$.AggCred. Finally, each voter uses original wallet public key pk_{v} and mvk to randomize the signature $\sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ as $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ through $\mathcal{TIAC}$.PrCred. $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ will be used as an anonymous credential in the election.

3.2.2.2 Election

During the election stage, the IPFS function serves the purpose of storing encrypted votes and its corresponding information in the form of an IPFS file. Once the voter confirms the correct storage of the file, the TransVote function transacts the address/digest of that file with the randomized credential to the blockchain. Subsequently, the smart contract executes the BlockVote function to validate the randomized credential of the voter, thereby enabling the inclusion of the signed valid transaction into the blockchain upon this validation.

- $\mathsf{IPFS}(\mathsf{Vote}_{\mathsf{v}}, \mathsf{mpk}, \sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})) \rightarrow \mathsf{StoredIPFS}_{\mathsf{v}}$: It is executed between the voters and IPFS providers. In the voter's part, each voter encrypts his vote $\mathsf{Vote}_{\mathsf{v}}$ with mpk and outputs the ciphertext as $\mathsf{ct}_{\mathsf{v}} = (\mathsf{C}_1, \mathsf{C}_2)$ through $\mathcal{HE}$.Enc. In this scenario, encryption requires two $\mathcal{ZKP}$: vote validity proof (VV_{v}) to prove the message is in a certain range, and vote sum proof (VS_{v}) to prove no more than one candidate selected. In the IPFS part, IPFS takes (ct_{v},VV_{v}, VS_{v}, $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$), then generates digest of the file stored in IPFS as $\mathsf{StoredIPFS}_{\mathsf{v}}$ to ensure correct storage.
- $\mathsf{TransVote}(\mathsf{StoredIPFS}_{\mathsf{v}}, \mathsf{sk}_{\mathsf{v}}, \mathsf{mvk}, \mathsf{pk}_{\mathsf{v}}, \sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})) \rightarrow \mathsf{Tx}_{\mathsf{v}}$: It is run by the voter's device to transact the vote. If the stored data in IPFS ($\mathsf{StoredIPFS}_{\mathsf{v}}$) is correct, then the voter digitally signs this IPFS address with his signing key sk_{v} as $\mathsf{t1}_{\mathsf{v}}$ via $\mathcal{DS}$.Sign. Also, the voter generates new randomized credential as $\mathsf{t2}_{\mathsf{v}}$ using $\mathsf{mvk}, \mathsf{pk}_{\mathsf{v}}, \sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ via $\mathcal{TIAC}$.PrCred. In total, blockchain transaction Tx_{v} consists of a digitally signed IPFS address ($\mathsf{t1}_{\mathsf{v}}$) and voter's randomized credential $\mathsf{t2}_{\mathsf{v}}$. This additional randomized credential in the IPFS address will enable

anyone to verify that tallying results contain only the votes coming from the eligible voters.

- $\mathsf{BlockVote}(\mathsf{Tx_v}, \mathsf{mvk}) \rightarrow \mathsf{Tx_{ID}}$: The block generation of vote function is run by Smart Contract (SC). It checks the proof π_v in randomized credential and the signature on credential $\mathsf{t2_v}$ using mvk, then writes the signed transaction $\mathsf{t1_v}$ to the blockchain and sends blockchain transaction ID $(\mathsf{Tx_{ID}})$ to the voter.

3.2.2.3 Post-Election

In the post-election stage, the AggVote function performs the aggregation of the encrypted votes. Subsequently, any set of t EA decrypts the aggregated vote partially employing the ParDecVote function. Finally, the smart contract executes the TallyVote function to obtain the complete result.

- $\mathsf{AggVote}(\mathsf{t1_v}, \mathsf{StoredIPFS_v}, \mathsf{pk_v}) \rightarrow \mathsf{AVote}$: The aggregation of the vote function is run by SC that takes all signed transactions $\mathsf{t1_v}$ in the blockchain. If the digital signature $\mathsf{t1_v}$ on $\mathsf{StoredIPFS_v}$, and $\mathcal{ZKP}$ of $\mathsf{ct_v}$ $(\mathsf{VV_v}, \mathsf{VS_v})$ are verified, ciphertexts are multiplied to generate aggregated votes $\mathsf{AVote} = (\mathsf{C'_1}, \mathsf{C'_2})$.

- $\mathsf{ParDecVote}(\mathsf{AVote}, \mathsf{sk_i}) \rightarrow (\mathsf{pd_i}, \mathsf{CD_i})$: The partial decryption vote function is run by any set of threshold (t) EA authority. Taking AVote from the AggVote function, and $\mathsf{sk_i}$ from EA_i, it returns partial decryption share on these votes as $\mathsf{pd_i} = (\mathsf{C'_1})^{\mathsf{sk_i}}$. In addition to this, $\{\mathsf{EA_i}\}_{i=1}^{t}$ publishes the correct decryption proof $\mathsf{CD_i}$ that proves the secret value $\mathsf{sk_i}$ in previously published $\mathsf{pk_i}$ is the same as the secret value used in $\mathsf{pd_i}$ to assure correct partial decryption.

- $\mathsf{TallyVote}(\mathsf{pd_i}, \mathsf{CD_i}) \rightarrow \mathsf{Result}$: TallyVote function is run by the SC after the execution of ParDecVote function. SC first verifies the correctness of decryption proof on $\mathsf{CD_i}$, then homomorphically decrypts the aggregated votes taking $\{\mathsf{pd_i}\}_{i=1}^{t}$ and $\mathsf{AVote}(= \mathsf{C'_1}, \mathsf{C'_2})$. In the detail of this homomorphic decryption, SC first computes $\mathsf{C'_2} \cdot \prod_{\mathsf{i}=1}^{\mathsf{t}} \mathsf{pd_i^{l_i}} = \mathsf{C'_2} \cdot \prod_{\mathsf{i}=1}^{\mathsf{t}} (\mathsf{C'_1})^{\mathsf{sk_i^{l_i}}} = \mathsf{C'_2} \cdot (\mathsf{C'_1})^{\mathsf{sk}}$ where $\mathsf{l_i}$ is the Lagrange coefficient, then employs a brute force method on this value, that is restricted to the number of voters, to publish Result of the election.

3.2.3 An Efficient System Instantiation of Proba

Distributed Key Generation. In the pre-election stage, the vote encryption key is generated by a fully distributed threshold non-interactive Pedersen Key Generation [53] as in a fully distributed variant of Helios [37]. Thus, the key generation setup of the election is done without a dealer and ensures that the election key is uniformly distributed. In that scheme, each EA builds their secret keys, and then the public key of the election is constituted with these secret keys. The verification keys of the EAs will later enable non-interactive partial decryption of the encrypted votes under the election public key. In that case, any set of administrators less than the threshold cannot learn any information on the encrypted votes.

Homomorphic Encryption. For the encryption, the El Gamal algorithm is chosen since election schemes based on homomorphic encryption, except for El Gamal, fail when there are more candidates for election [80]. Also, plaintext multiplication in multiplicative homomorphic algorithms induces overflow in plaintext range [88]. Thus, Proba continued to use the additive El Gamal called exponential El Gamal. The traditional El Gamal algorithm is made additively with Cramer transformation by encrypting g^m instead of message m for some generator g [40]. Also, at the end of the decryption, it additionally requires solving discrete logarithm problems. However, this additional step will not pose a problem because this number will be limited to the number of voters in a voting system.

Zero-Knowledge Proof. In Helios, there are three types of zero-knowledge proofs. These proofs are disjunctive Chaum-Pederson proofs [33], and they made noninteractive with Fiat-Shamir transformations [48]. For the intended ZKPs, Proba continued to utilize these proofs. ① The first zero-knowledge proof is required to prove that the corresponding encrypted vote encodes an integer between 0 and the candidate number (max). In a simple election with two candidates, this can be thought of as the encrypted vote encoded 0 or 1. In that case, in addition to the encrypted vote, the voter sends proof that the encrypted vote is within a certain allowed range. Thus, the voter can only cast up to one vote for a candidate. ② The second zero-knowledge proof is required to prove that the sum encrypted vote is 1. This means voters need to prove that the sum of their plaintexts corresponding to their ciphertexts is 1. Thus,

voters can choose up to one candidate. ③ The third proof is required to prove correct decryption. Contrary to the first two proofs, the last proof is produced by the authority that will decrypt the encrypted votes. Thus, it is proved that the decryption process is done correctly.

Anonymous Credential. In Proba, TIAC scheme is used to authorize the anonymous credential by signing it. Eligibility of the voter's wallet addresses is proved with the knowledge of signature on an anonymous wallet public key using Coconut/TIAC construction [129]. For the anonymous credential in threshold issuance, the improved Coconut structure [113] is chosen. In this case, while TIAC provides blindness, unforgeability, and unlinkability, the improved TIAC provides additional unconditional privacy that ensures the security of the system despite the adversary's unlimited computational power. It should be noted that all of the ZKPs necessary for the Coconut system are founded upon standard Sigma protocols that rely on the DH assumption. These proofs demonstrate knowledge of the discrete logarithm's representation.

Consortium Blockchain. The consortium blockchain type is used in Proba since it is managed by more than one authority; transactions are fast and have no cost. As a consortium platform, Java-based Hyperledger Besu [1] that is compatible with the most popular wallets like Metamask and suitable for business projects due to multiple authority management is chosen. Also, Besu allows using of various consensus protocols such as Proof of Stake, Proof of Work, and Proof of Authority (IBFT 2.0, QBFT, and Clique).

Consensus Algorithm. Another important issue to consider is the consensus algorithm that determines which blocks are added to the blockchain. As opposed to the computational effort in public consensus algorithm as Proof of work in Bitcoin, deterministic immutability is a more desirable requirement for consortium blockchain [117]. Since the nodes in the consortium blockchain know each other, contrary to the public blockchain with no level of trust between each other, Proof of Authority (PoA) can be used as a consensus. However, instead of using naive PoA, to provide balance in security and performance, leader-based Istanbul Byzantine Fault Tolerance (IBFT) which was developed by AMIS Technologies [93] can be chosen.

[1] https://www.hyperledger.org/use/besu

The formulation of all election phases in Proba can be seen in Figure 3.3, 3.4, and 3.5. In these figures, the ZKP formulas are not provided because of their large size. However, further details on proof of knowledge of discrete logarithm PoK in pre-election, vote validity VV proof (called disjunctive proof of plaintext equality) - vote sum VS proof (called proof of inequality of two discrete logarithms) in the election, and correct decryption CD proof (called proof of the equality of two discrete logarithms) in post-election can be found on [64]. Also, since the signature algorithm depends on the underlying blockchain platform, elliptic curve digital signatures can be used as a default.

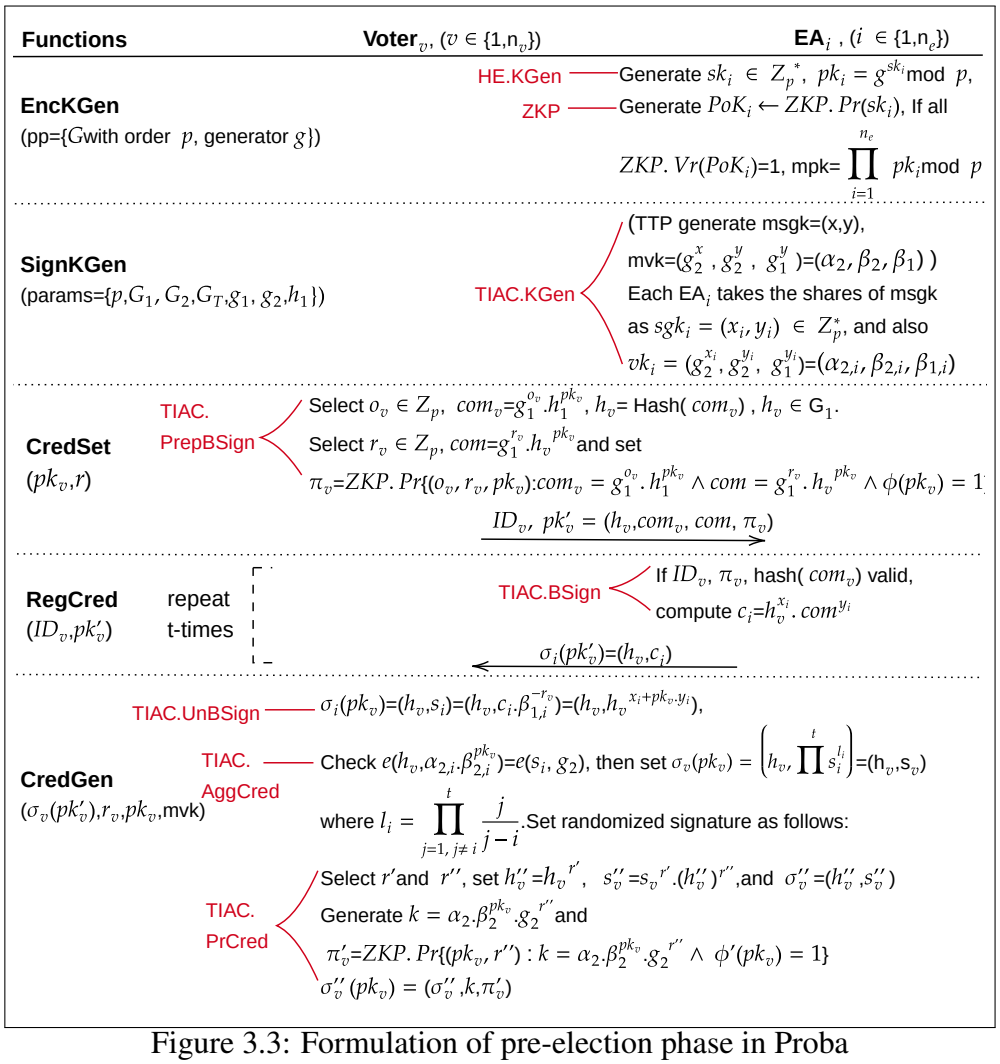

Figure 3.3: Formulation of pre-election phase in Proba

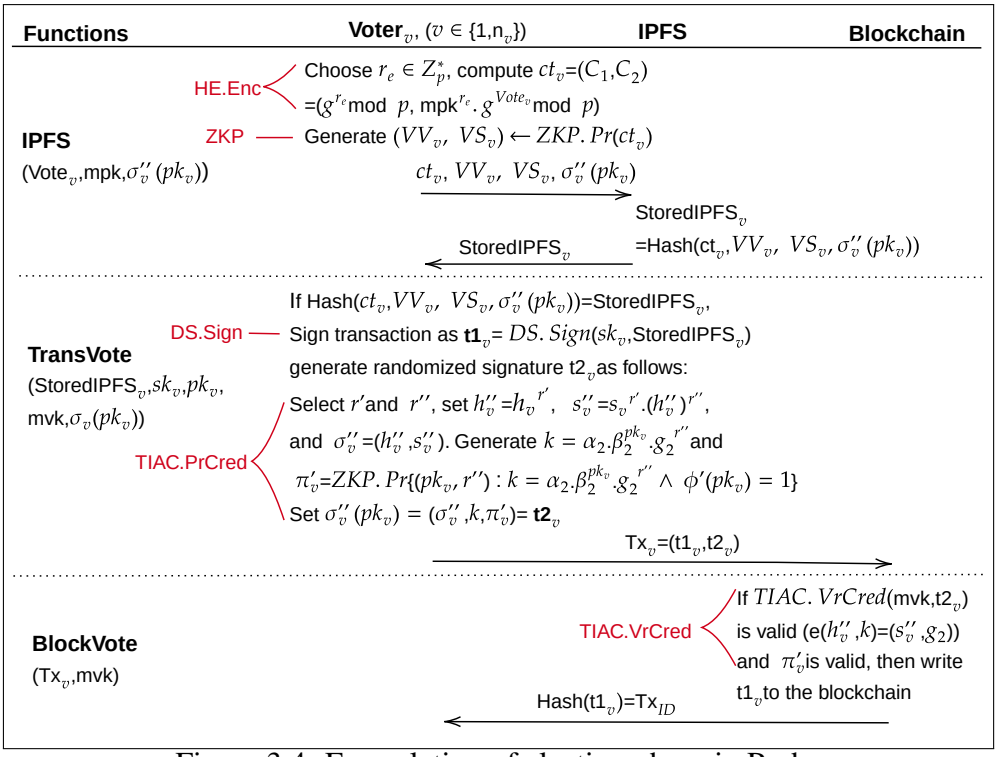

Figure 3.4: Formulation of election phase in Proba

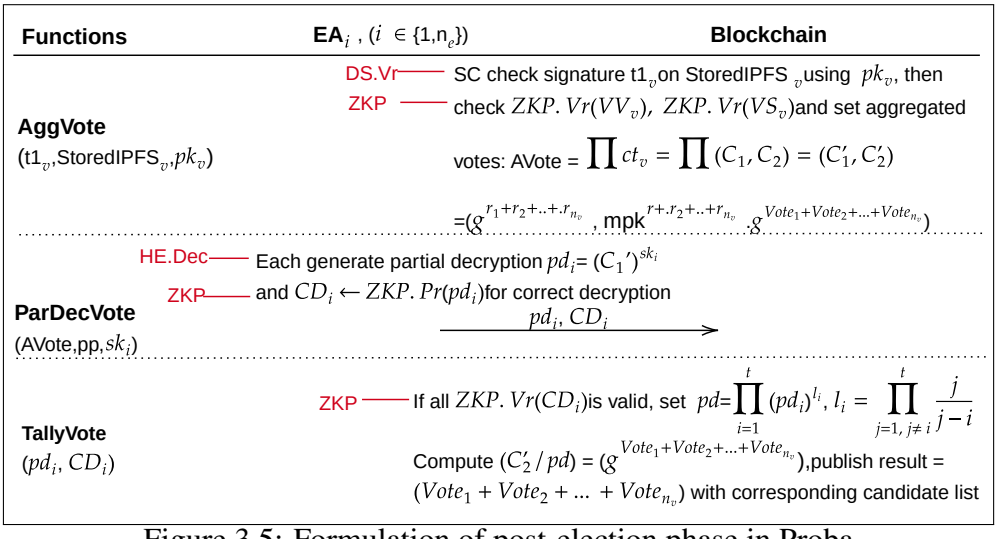

Figure 3.5: Formulation of post-election phase in Proba

3.3 System Analysis

3.3.1 Security Analysis

The initial analysis of any voting system requires an evaluation of the basic requirements of the voting system. Since the modifications in Blockchain-powered Helios strengthen the Proba system in terms of robustness, and privacy we start with these formal security definitions. To achieve robustness and privacy, Proba used a $\mathcal{TIAC}$ scheme, thereby the modified stages also require a formal definition of eligibility and verifiability. This is because the modified stages affect the voter's ability to access the election through credentials and verify their vote.

Oracles. Our formal security definition relies on some oracles that Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ has access to them through challenger $\mathcal{C}$.

- $\mathcal{O}_{\mathsf{RegCred}}(\mathsf{pk}'_{\mathsf{v}}, \mathsf{ID}_{\mathsf{v}})$: If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{RegCred}}$, valid partial signature on blind wallet public key $\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}})$ is given to $\mathcal{A}$ and it is also added to the registered credential list $\mathcal{L}_{\mathsf{cred}}$.

- $\mathcal{O}_{\mathsf{StoredVote}}(\mathsf{ct}_{\mathsf{v}}, \mathsf{auxinfo})$: If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{StoredVote}}$, the hash of encrypted vote ct_{v} and the related auxiliary information $\mathsf{auxinfo}$ ($\mathsf{t2}_{\mathsf{v}} = \sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}}), \mathsf{VV}_{\mathsf{v}}, \mathsf{VS}_{\mathsf{v}}$) are given to $\mathcal{A}$ as $\mathsf{StoredIPFS}_{\mathsf{v}}$ and it is also saved in the blockchain as valid the ballot list $\mathcal{L}_{\mathsf{bal}}$.

3.3.1.1 Eligibility

Eligibility is defined as the authorization to vote, and this authorization is provided by anonymous credentials on blind signatures in the Proba system. The concept of the blindness property entails the situation where the attacker is given two signatures on two blind messages in uncertain order, and then he tries to guess the correct order. Following this definition, eligibility is satisfied if for all adversaries $\mathcal{A}$, the advantage on winning the eligibility experiment defined in Figure 4.22 is negligible, i.e.,

$Adv_{\mathcal{A}}^{elig}(\lambda) = Pr[\mathsf{Exp}_{\mathcal{A}}^{elig,b}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$

$\underline{\mathsf{Exp}_{\mathcal{A}}^{elig,b}(\lambda)}$
$\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda)$:
$(\mathsf{pk}'_0, \mathsf{pk}'_1) \leftarrow \mathcal{A}^{\mathcal{O}_{Choose}}(\mathsf{params})$:
For $\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\sigma_{\mathsf{i}}(\mathsf{pk}'_0) = \mathsf{RegCred}(\mathsf{pk}'_0, \mathsf{ID}_0)$,
$\sigma_{\mathsf{i}}(\mathsf{pk}'_1) \xleftarrow{\$} S$
if b=0:
return b′ such that $\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{b}}) \in \mathcal{L}_{\mathsf{cred}}$
if b=1:
return b′ such that $\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{b}}) \notin \mathcal{L}_{\mathsf{cred}}$

Figure 3.6: Eligibility experiment for Proba

Let the registered credential output be defined over the signature space S. An eligibility experiment ($\mathsf{Exp}_{\mathcal{A}}^{elig,b}$) between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$ runs as follows:

- $\mathcal{C}$ generates the public paramaters params for signature scheme, then publish it to adversary $\mathcal{A}$.
- $\mathcal{A}$ chooses two blind wallet public keys $\mathsf{pk}'_0, \mathsf{pk}'_1$ and sends them to $\mathcal{C}$.
- $\mathcal{C}$ computes the signature for the credential as follows:
  - if $\mathsf{b} = 0$; $\sigma_{\mathsf{i}}(\mathsf{pk}'_0) \leftarrow \mathsf{RegCred}(\mathsf{ID}, \mathsf{pk}'_0)$, then $\sigma_{\mathsf{i}}(\mathsf{pk}'_0)$ is added to the registered credential list $\mathcal{L}_{\mathsf{cred}}$
  - if $\mathsf{b} = 1$; $\sigma_{\mathsf{i}}(\mathsf{pk}'_1) \xleftarrow{\$} \mathsf{S}$

  then sends it to $\mathcal{A}$

$\mathcal{A}$ wins the game if he distinguishes the real blind signature $\sigma_{\mathsf{v}}(\mathsf{pk}'_0)$ from a random blind signature $\sigma_{\mathsf{v}}(\mathsf{pk}'_1)$, thus outputs the corresponding blind public key declaring the signature on it is in the eligible credential set or not.

Theorem 1. *Proba provides eligibility if the TIAC signature on the voter's credential is blind.*

Proof. In particular, for every eligibility adversary $\mathcal{A}$ that attacks the Proba system as in Figure 4.22, there exists a $\mathcal{TIAC}$ adversary $\mathcal{B}$ that attacks blindness property of anonymous $\mathcal{TIAC}$ credential. According to Figure 4.22, $\mathcal{A}$ who gains a nonnegligible advantage for eligibility, can distinguish an eligible blind signature from a random blind signature. We will construct another adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine for some appropriate input and plays the role of challenger to $\mathcal{A}$. According to blindness, $\mathcal{B}$, who gains a nonnegligible advantage for blindness, can distinguish the order of the blind signatures.

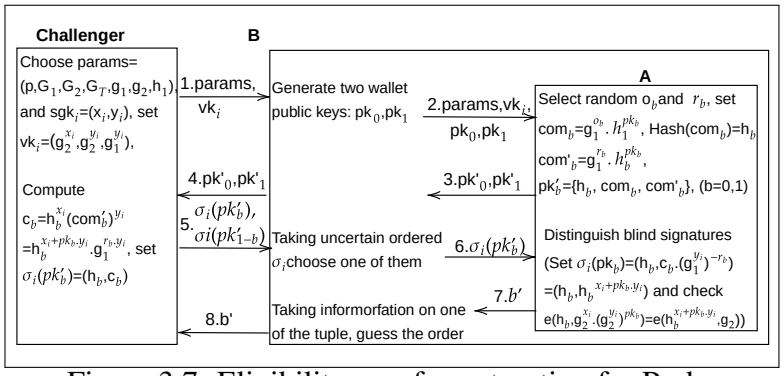

Figure 3.7: Eligibility proof construction for Proba

In Figure 3.7, $\mathcal{B}$'s challenger $\mathcal{C}$ chooses params then set signing key $\mathsf{sgk_i} = (\mathsf{x}, \mathsf{y})$ and its corresponding verification key $\mathsf{vk_i} = (\mathsf{g_2^{x_i}}, \mathsf{g_2^{y_i}}, \mathsf{g_1^{y_i}})$. $\mathcal{C}$ sends $\mathsf{vk_i}$ to $\mathcal{B}$. $\mathcal{B}$ generates his wallet public keys $\mathsf{pk_0}, \mathsf{pk_1}$ based on the underlying blockchain platform and sends them to $\mathcal{A}$ together with params and $\mathsf{vk_i}$. $\mathcal{A}$ makes the public wallet keys blind as $\mathsf{pk'_0}, \mathsf{pk'_1}$ and $\mathcal{B}$ forwards them to $\mathcal{C}$. $\mathcal{C}$ runs the $\mathcal{TIAC}$.BSign in RegCred function to sign them blindly as $\sigma_\mathsf{i}(\mathsf{pk'_0}), \sigma_\mathsf{i}(\mathsf{pk'_1})$. Taking $\sigma_\mathsf{i}(\mathsf{pk'_0}), \sigma_\mathsf{i}(\mathsf{pk'_1})$ in uncertain order from $\mathcal{C}$, $\mathcal{B}$ sends one of the signature $\sigma_\mathsf{i}(\mathsf{pk'_b})$ to $\mathcal{A}$. Since $\mathcal{A}$ has a nonnegligible advantage over eligibility, he can distinguish blind signatures and outputs its corresponding $\mathsf{b'}$ to $\mathcal{B}$. Taking this information from $\mathcal{A}$, $\mathcal{B}$ can distinguish the order of the signature taken from $\mathcal{C}$ with nonnegligible advantage.

□

3.3.1.2 Privacy

Privacy is the removal of the link between the voter and the vote. This is achieved in Proba through the utilization of the exponential El-Gamal algorithm that preserves the

vote privacy and the $\mathcal{TIAC}$ scheme which protects the privacy of voters by introducing randomness to the anonymous credential. Privacy is satisfied if for all adversaries $\mathcal{A}$ the advantage on winning the security experiment defined in Figure 3.8 and Figure 3.10 are negligible, i.e.:

$$Adv_{\mathcal{A}}^{\mathsf{priv1}}(\lambda) = \left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{priv1},1}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{priv1},0}(\lambda) = 1]\right|$$

$$Adv_{\mathcal{A}}^{\mathsf{priv2}}(\lambda) = \left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{priv2},1}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{priv2},0}(\lambda) = 1]\right|$$

$\mathsf{Exp}_{\mathcal{A}}^{\mathsf{priv1,b}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$,
$\leftarrow \mathsf{EncKGen}(\mathsf{pp})$:
$(\mathsf{Vote}_0, \mathsf{Vote}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Choose}}}(\mathsf{pp})$
For $\mathsf{b} \xleftarrow{\$} \{0,1\}$ and
$\mathsf{ct}_{\mathsf{b}} \leftarrow \mathcal{HE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{Vote}_{\mathsf{b}})$:
$\mathsf{b}' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Guess}}}(\mathsf{ct}_{\mathsf{b}})$
if $\mathsf{b}' = \mathsf{b}$
return 1
else return 0

Figure 3.8: Experiment for Vote Privacy in Proba

Vote privacy attack game. This game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$ runs as follows:

- $\mathcal{C}$ generates the master public key $\mathsf{mpk} = \mathsf{g}^{\mathsf{x}}$ for encryption scheme based on public parameters pp, and publish it to adversary $\mathcal{A}$.
- $\mathcal{A}$ choose two distinct but the same length messages Vote_0 and Vote_1, then sends them to the challenger $\mathcal{C}$.
- $\mathcal{C}$ selects uniformly random bit $\mathsf{b} \xleftarrow{\$} \{0,1\}$, then sends challenge ciphertext $\mathsf{ct}_{\mathsf{b}} \leftarrow \mathcal{HE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{Vote}_{\mathsf{b}})$ to $\mathcal{A}$ where $\mathsf{ct}_{\mathsf{b}} = (\mathsf{g}^{\mathsf{y}}, \mathsf{g}^{\mathsf{x.y}+\mathsf{Vote}_{\mathsf{b}}}) = (c_1, c_2)$
- $\mathcal{A}$ outputs a guess bit $\mathsf{b}' \in \{0,1\}$

$\mathcal{A}$ wins the game if $\mathsf{b}' = \mathsf{b}$.

Remark 1. Instead of a bit-guessing experiment of this attack game, the game can also be designed with two ciphertexts, one of which is a random element in ciphertext space and the other of which is a real encrypted vote. However, this game is weaker in terms of Challenger $\mathcal{C}$.

Remark 2. Vote privacy game ensures vote indistinguishability which means it is infeasible to distinguish pairs of ciphertexts based on plaintexts better than $\frac{1}{2}$ probability [57]. Thus, it is the same as the semantic security (SS) game in Definition 5 for El-Gamal algorithm. However, to be more clear, we present its proof here.

Theorem 2. *Proba preserves vote privacy if the DDH assumption holds in* $\mathbb{G}$.

Proof. Assuming there exists an adversary $\mathcal{A}$ that breaks the vote privacy game, we can construct an adversary $\mathcal{B}$ who tries to break the DDH assumption in $\mathbb{G}$. Since the encryption algorithm in Proba uses Exponential El-Gamal, we can take the ciphertext $\mathsf{ct_b}$ on message $\mathsf{Vote_b}$ as $(\mathsf{g^y}, (\mathsf{g^x})^{\mathsf{y}}.\mathsf{g}^{\mathsf{Vote_b}})$ where g is a generator of $\mathbb{G}$, $\mathsf{g^x}$ is the public key, and y is uniformly chosen random value.

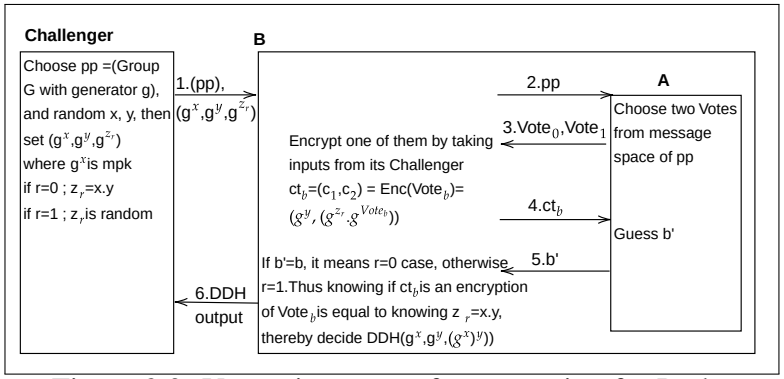

Figure 3.9: Vote privacy proof construction for Proba

In Figure 3.9, $\mathcal{C}$ sets up public parameters pp then choose random x and y. Subsequently, $\mathcal{C}$ generates DDH inputs as $(\mathsf{mpk} = \mathsf{g^x}, \mathsf{g^y}, \mathsf{g^{z_r}})$, then sends them to $\mathcal{B}$. In these inputs, if $\mathsf{r} = 0$, $\mathsf{g^{z_r}} = \mathsf{g^{x \cdot y}}$; if $\mathsf{r} = 1$, $\mathsf{g^{z_r}}$ is randomly chosen value. $\mathcal{B}$ forwards pp to $\mathcal{A}$ so that $\mathcal{A}$ can choose two votes from the message space of the encryption algorithm. Taking two votes from $\mathcal{A}$, $\mathcal{B}$ encrypts one of them by using the inputs taken from $\mathcal{C}$ as $\mathsf{ct_b} = (\mathsf{g^y}, \mathsf{g^{z_r}}.\mathsf{g}^{\mathsf{Vote_b}})$. According to the nonnegligible advantage of the vote privacy game, we assume that $\mathcal{A}$ can distinguish which encrypted vote $\mathsf{ct_b}$ belongs to which message $\mathsf{Vote_b}$ or if $\mathsf{ct_b}$ doesn't belong to any encryption (the case of random $\mathsf{z_r}$). Based on the response of $\mathcal{A}$, $\mathcal{B}$ can decide the DDH tuple. In the event that $\mathcal{A}$

produces the output b', which is equivalent to b, $\mathcal{B}$ is aware that this corresponds to the scenario where r is equal to zero. Using this information, $\mathcal{B}$ can decide whether the input (g^x, g^y, g^{z_r}) is a DDH tuple or random tuple with half the advantage that $\mathcal{A}$ has.

□

$\mathsf{Exp}^{\mathsf{priv2},b}_{\mathcal{A}}(\lambda)$
$\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda)$:
$(\mathsf{pk}_0, \mathsf{pk}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Choose}}}(\mathsf{params})$
For $\mathsf{b} \xleftarrow{\$} \{0,1\}$ and
$\mathsf{pk}'_\mathsf{b} \leftarrow \mathsf{CredSet}(\mathsf{pk}_\mathsf{b}, \mathsf{r}_\mathsf{b})$,
$\sigma_\mathsf{v}(\mathsf{pk}'_\mathsf{b}) \leftarrow \mathsf{RegCred}(\mathsf{ID}_\mathsf{v}, \mathsf{pk}'_\mathsf{b})$,
$\sigma''_\mathsf{v}(\mathsf{pk}_\mathsf{b}) \leftarrow \mathsf{CredGen}(\sigma_\mathsf{v}(\mathsf{pk}'_\mathsf{b}), \mathsf{r}_\mathsf{b}, \mathsf{pk}_\mathsf{b}, \mathsf{mvk})$:
$\mathsf{b}' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Guess}}}(\sigma''_\mathsf{v}(\mathsf{pk}_\mathsf{b}))$
if $\mathsf{b}' = \mathsf{b}$
return 1
else return 0

Figure 3.10: Experiment for Voter Privacy in Proba

Voter Privacy Attack Game. This game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$ runs as follows:

- $\mathcal{C}$ generates the public parameters params for signature scheme and publish it to adversary $\mathcal{A}$.
- $\mathcal{A}$ chooses two distinct but the same length messages pk_0 and pk_1, then sends them to $\mathcal{C}$.
- $\mathcal{C}$ uses $\mathsf{CredSet}$, $\mathsf{RegCred}$, and $\mathsf{CredGen}$ functions to get the randomized signature $\sigma''_\mathsf{v}(\mathsf{pk}_\mathsf{b})$ on one of the messages, then sends it to $\mathcal{A}$.
- $\mathcal{A}$ outputs a guess bit $\mathsf{b}' \in \{0,1\}$

$\mathcal{A}$ wins the game if he guesses correctly b' s.t. $\mathsf{b} = \mathsf{b}'$

Remark 3. Instead of a bit-guessing experiment of this attack game, the game can also be designed with an indistinguishability approach where one credential is a random

element and the other is a real randomized signature. However, this game is weaker in terms of Challenger $\mathcal{C}$.

Theorem 3. *Proba preserves voter privacy under XDH assumption.*

Proof. Let adversary $\mathcal{A}$ breaks the voter privacy game with nonnegligible probability, then we can construct an adversary $\mathcal{B}$ who breaks the XDH assumption.

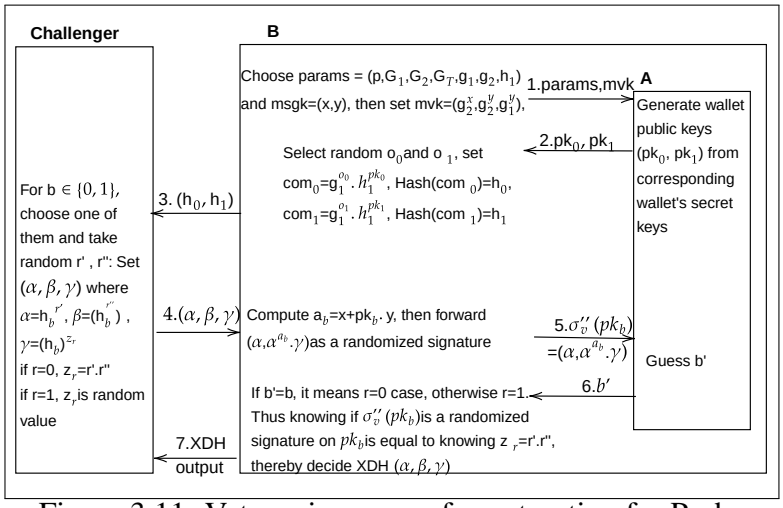

Figure 3.11: Voter privacy proof construction for Proba

In Figure 3.11, $\mathcal{B}$ chooses public parameters params and master signing key msgk, then set master verification key mvk. $\mathcal{A}$ takes params and mvk from its $\mathcal{C}$, then derives the public key of two wallets by utilizing the secret key of each wallet, which is determined by the underlying blockchain platform. Taking two wallet's public keys from $\mathcal{A}$, $\mathcal{B}$ generates the commitment on both of the public keys and sends the hash of their commitments h_0, h_1 to $\mathcal{C}$ as generated in $\mathcal{TIAC}$.PrepBSign. $\mathcal{C}$ chooses one of the h_b value ($\mathsf{b} \in \{0,1\}$), and chooses random r', r''. Then $\mathcal{C}$ computes the value $\alpha = \mathsf{h}_\mathsf{b}^{\mathsf{r}'}$, $\beta = \mathsf{h}_\mathsf{b}^{\mathsf{r}''}$, $\gamma = \mathsf{h}_\mathsf{b}^{\mathsf{z_r}}$ and sends (α, β, γ) to $\mathcal{B}$. In these inputs, if $\mathsf{r} = 0$, $\mathsf{z_r} = \mathsf{r}' \cdot \mathsf{r}''$; if $\mathsf{r} = 1$, $\mathsf{z_r}$ is randomly chosen different value. $\mathcal{B}$ computes $a_b = x + pk_b \cdot y$ and forwards $(\alpha, \alpha^{\mathsf{a_b}}.\gamma)$ to $\mathcal{A}$ as a randomized signature on one of the public keys. Since $\mathcal{A}$ wins the game, he can correctly guess b. If $\mathcal{A}$ outputs b' where $\mathsf{b}' = \mathsf{b}$, then $\mathcal{B}$ knows that this is equal to $\mathsf{r} = 0$ case ($\mathsf{z_r} = \mathsf{r}'.\mathsf{r}''$). Thus, $\mathcal{B}$ can decide whether the input tuple is an XDH tuple or a random tuple with $\frac{1}{4}$ probability.

□

3.3.1.3 Verifiability

According to the election verifiability definition, voters should trust the election results without relying on some authority. Proba achieves this since the voter can trace his vote in the blockchain with collision-resistant hash functions and can detect such malicious behavior. Verifiability for the election is satisfied if for all adversaries $\mathcal{A}$ the advantage on winning the security game defined in Figure 3.12 is negligible, i.e.:

$$Adv_{\mathcal{A}}^{\mathsf{ver}}(\lambda) = Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ver}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

$\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ver}}(\lambda)$

$\mathsf{mpk} \leftarrow \mathsf{EncKGen}(\mathsf{pp})$:

$\mathsf{ct_v} \leftarrow \mathcal{HE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{Vote_v})$

$\mathsf{StoredIPFS_v} \leftarrow \mathcal{A}^{O_{\mathsf{StoredVote}}}(\mathsf{ct_v}, \mathsf{auxinfo})$

Output a new ciphertext $\mathsf{ct_v^*} \leftarrow \mathcal{HE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{Vote_v^*})$

$\mathsf{StoredIPFS_v^*} \leftarrow \mathcal{A}^{O_{\mathsf{StoredVote}}}(\mathsf{ct_v^*}, \mathsf{auxinfo})$:

return $\mathsf{StoredIPFS_v^*} = \mathsf{StoredIPFS_v}$

Figure 3.12: Verifiability game for Proba

Verifiability game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$ runs as follows:

- $\mathcal{C}$ generates master public key mpk for encryption, then publish it to adversary $\mathcal{A}$.
- Taking mpk from $\mathcal{C}$, $\mathcal{A}$ generates an encrypted vote as $\mathsf{ct_v}$ on plain vote $\mathsf{Vote_v}$.
- $\mathcal{A}$ queries the oracle $\mathcal{O}_{StoredVote}$ on input ciphertext $\mathsf{ct_v}$, as well as auxiliary information $\mathsf{auxinfo}$ which encompasses the randomized credential $\mathsf{t2_v}$ and the $\mathcal{ZKP}$ of $\mathsf{ct_v}$. This oracle saves the hash of the given input to the blockchain as $\mathsf{StoredIPFS_v} \in \mathcal{L}_{\mathsf{bal}}$.
- After some query, $\mathcal{A}$ produces a new ballot $\mathsf{StoredIPFS_v^*}$ that is not produced by the oracle.

$\mathcal{A}$ wins the game if $\mathsf{StoredIPFS_v^*} = \mathsf{StoredIPFS_v}$. According to this game, even if the voter has an eligible credential, the adversary can forge $\mathsf{StoredIPFS_v}$ that stores the

hashes of encrypted votes and other related auxiliary information, thus changing vote choice.

Theorem 4. *Proba satisfies verifiability under the DL assumption.*

Proof. The verifiability game in Figure 3.12 is based on the collision resistance property of hash functions. Collision resistance means that it is difficult to find two different inputs that hash to the same output such that $H(a) = H(b)$ and $a \neq b$, thus means for all efficient adversary $\mathcal{A}$, its advantage on collision-finding is negligible. In Proba, when the credential $\mathsf{t2_v} = \sigma''_{\mathsf{v}}(\mathsf{pk_v})$ is valid, an encrypted vote ct_v is added to the blockchain. In fact, instead of the encrypted vote and its auxiliary information together, the Secure Hash Algorithm Standard-based IPFS address ($\mathsf{StoredIPFS_v}$) is recorded on the blockchain.

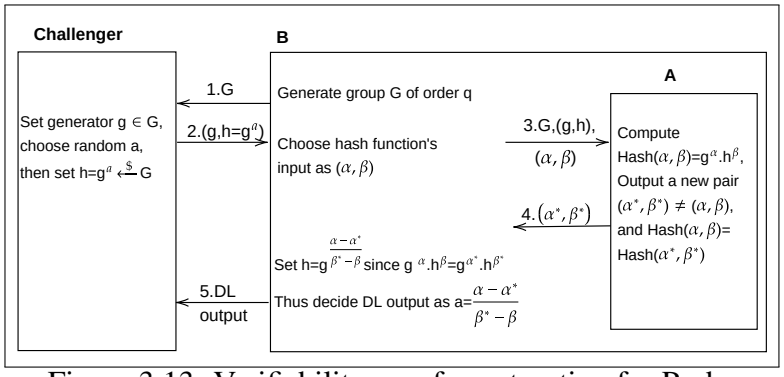

Figure 3.13: Verifiability proof construction for Proba

In Figure 3.13, $\mathcal{B}$ generates a cyclic group $\mathbb{G}$ of prime order q and send it to $\mathcal{C}$. $\mathcal{C}$ computes a generator g $\in\mathbb{G}$, and also set $\mathsf{h} = \mathsf{g^a}$ as an arbitrary group element in G by choosing random a. Taking g, h from $\mathcal{C}$, $\mathcal{B}$ chooses an input of hash function as α, β and sends these together to $\mathcal{A}$. $\mathcal{A}$ computes the hash value according to the taken input. Let collision-resistant hash function (CRHF) be constructed as $Hash(\alpha, \beta) = g^{\alpha} \cdot h^{\beta}$ mod q. $\mathcal{A}$ finds a collision $Hash(\alpha, \beta) = Hash(\alpha^*, \beta^*)$, $(\alpha, \beta) \neq (\alpha^*, \beta^*)$ with nonnegligible advantage and sends it to $\mathcal{B}$. In this case, $\mathcal{B}$ constructs $\mathsf{h} = \mathsf{g}^{\frac{\alpha-\alpha^*}{\beta^*-\beta}}$ since $\mathsf{g}^{\alpha} \cdot \mathsf{h}^{\beta} = \mathsf{g}^{\alpha^*} \cdot \mathsf{h}^{\beta^*}$. Thus, $\mathcal{B}$ breaks the DL assumption by outputting $a = \frac{\alpha-\alpha^*}{\beta^*-\beta}$.

□

3.3.1.4 Robustness

Robustness means resistance to malicious behavior. While the blockchain ensures robustness in the election phase of Proba, the absence of any interaction with the blockchain in the pre-election phase necessitates an additional precaution to ensure the robustness of the system. Proba system takes this precaution by using $\mathcal{TIAC}$ scheme, thus allowing for the issuing credential procedure to be distributed in pre-election. Robustness in Proba is satisfied if for all adversaries $\mathcal{A}$ the advantage on winning the security game defined in Figure 3.14 is negligible, i.e.:

$$Adv_{\mathcal{A}}^{\mathsf{rob}}(\lambda) = \left|\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{rob},r}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

$\mathsf{Exp}_{\mathcal{A}}^{\mathsf{rob},r}(\lambda)$

$\mathsf{mvk} \leftarrow \mathsf{SignKGen}(\mathsf{params})$:

For $\mathsf{i} = 1$ to $\mathsf{t}' < \mathsf{t}$:

$\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RegCred}}}(\mathsf{ID}_{\mathsf{v}}, \mathsf{pk}'_{\mathsf{v}})$

$\sigma_{\mathsf{i}}(\mathsf{pk}_{\mathsf{v}}) \leftarrow \mathcal{TIAC}.\mathsf{UnBSign}(\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}}), \mathsf{r})$

Then generate new $\{\sigma_{\mathsf{j}}(\mathsf{pk}_{\mathsf{v}})_{\mathsf{j}=\mathsf{t}'+1}^{\mathsf{t}}\} \leftarrow \mathsf{S}'$

$\sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}}) \leftarrow \mathcal{TIAC}.\mathsf{AggCred}(\{\sigma_{\mathsf{i}}(\mathsf{pk}_{\mathsf{v}})\}_{\mathsf{i}=1}^{\mathsf{t}'} \wedge \{\sigma_{\mathsf{j}}(\mathsf{pk}_{\mathsf{v}})_{j=t'+1}^{t}\})$

$\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}}) \leftarrow \mathcal{TIAC}.\mathsf{PrCred}(\sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}}), \mathsf{mvk}, \mathsf{pk}_{\mathsf{v}})$:

return $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ s.t. $\mathcal{TIAC}.\mathsf{VrCred}(\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}}), \mathsf{mvk}) = 1$

Figure 3.14: Robustness game for Proba

Let the partial credential output $\sigma_{\mathsf{i}}(\mathsf{pk}_{\mathsf{v}})$ be defined over the signature space S'. Robustness game between the challenger $\mathcal{C}$ and adversary $\mathcal{A}$ runs as follows:

- $\mathcal{C}$ generates master verification key mvk for signature, then publish it to adversary $\mathcal{A}$.
- $\mathcal{A}$ makes $\mathsf{t}' < \mathsf{t}$ many queries $\mathcal{O}_{\mathsf{RegCred}}$ and gets blinded signatures as $\{\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}})\}_{i=1}^{t'}$.
- $\mathcal{A}$ removes the blind factor in each $\{\sigma_{\mathsf{i}}(\mathsf{pk}'_{\mathsf{v}})\}_{\mathsf{i}=1}^{\mathsf{t}'}$ using $\mathcal{TIAC}.\mathsf{UnBSign}$ and gets the original partial signatures $\{\sigma_{\mathsf{i}}(\mathsf{pk}_{\mathsf{v}})\}_{\mathsf{i}=1}^{\mathsf{t}'}$.
- Then $\mathcal{A}$ generates the remaining new signatures on pk_{v} by choosing random partial signatures $\{\sigma_{\mathsf{j}}(\mathsf{pk}_{\mathsf{v}})\}_{\mathsf{j}=\mathsf{t}'+1}^{\mathsf{t}}$ from signature space S'.

- Subsequently, $\mathcal{A}$ uses $\mathcal{TIAC}.\mathsf{AggCred}$ to generate single credential $\sigma_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ and $\mathcal{TIAC}.\mathsf{PrCred}$ to generate randomized credential $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$.

$\mathcal{A}$ wins the game if the generated credential $\sigma''_{\mathsf{v}}(\mathsf{pk}_{\mathsf{v}})$ is valid in $\mathcal{TIAC}.\mathsf{VrCred}$.

Theorem 5. *Proba satisfies robustness if the TIAC signature on the voter's credential captures one-more unforgeability*

Proof. Considering one-more unforgeability in TIAC, an adversary queries l-many times TIAC signing oracle for different blind messages $(\mathsf{pk}'^{(1)}_{\mathsf{v}}, \mathsf{pk}'^{(2)}_{\mathsf{v}}, \ldots, \mathsf{pk}'^{(\mathsf{l})}_{\mathsf{v}})$ on the same wallet public key pk_{v} through the Pedersen commitment, then takes blind signature on them. The adversary is then cannot forge at least one valid signature on another blind message $\mathsf{pk}'^{(\mathsf{l}+1)}_{\mathsf{v}}$ on the same wallet public key pk_{v}.

Assuming there exists an adversary $\mathcal{A}$ who breaks the robustness game in Figure 3.14, we can construct an adversary $\mathcal{B}$ who tries to break one-more unforgeability as defined in Definition 6. Breaking the robustness game means that an adversary $\mathcal{A}$ has collected $\mathsf{t}' < \mathsf{t}$ signatures through signing oracle, and he has the ability to produce the remaining signatures required in $\mathcal{TIAC}.\mathsf{AggCred}$ without interacting with signing oracle. In this case, $\mathcal{B}$ directly breaks one-more unforgeability. This is because $\mathcal{B}$ is also capable of generating a number of valid signatures greater than t' through engaging in the signing protocol t' times. □

The rest of the security requirements will be given informally. The discussed security requirements prove that the generic construction for Proba in Algorithm 1 is secure.

- **Uniqueness.** Since Proba allows one voter to register one public wallet key, thus satisfies uniqueness. Even if more than one vote is sent to the system through the same public wallet key, only the last vote for each voter is counted during the counting phase.
- **Fairness.** In Proba, the encryption key is created in a distributed manner between the EAs. Even if EA holds a part of the secret encryption key for election, the main secret encryption key will not be compromised due to the threshold value. Thus intermediate results are not revealed before the election is closed, and the system satisfies fairness.

- **Integrity:** Since Proba is based on blockchain technology, encrypted votes are seen as block transactions. Also, since each blockchain transaction has transaction hash/ID, there cannot be an unauthorized change in the votes. Thus integrity is satisfied in this blockchain-based system.

- **Transparency:** In the consortium blockchain that is used in Proba, direct access to data and transaction submissions are restricted to its specific participants. However, voters can access all transaction and see all phases. Thus, Proba is transparent for its voters.

- **Coercion-resistance.** Proba is suitable for low coercion elections as in the Helios system. In Proba, even if the voters are under pressure, they have the right to vote again, but since voters choose their blinding factor in $\mathcal{TIAC}$.PrepBSign and randomization factor in $\mathcal{TIAC}$.PrCred, the system is not coercion-resistant if the voter reveals this value.

- **Receipt-freeness.** After the encrypted vote is written to the blockchain, the transaction ID is given to the voter. With this receipt, encrypted votes kept in the IPFS system, proofs showing the validity of the votes, and proofs proving the eligibility of the randomized signature on credential can be viewed. In this case, while transaction ID provides verifiability, it does not provide receipt-freeness.

3.3.2 Performance Analysis

Since efficiency is also an important criterion for the systems, this section analyzes the efficiency of Proba. The implementations have been done in Python, using the hashlib library for the non-interactive zero-knowledge proofs. Also, it has been run into Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz. In the implementation, a 4096-bits security for Exponential El Gamal encryption and 256-bits security for $\mathcal{TIAC}$ are used. After the key generation setup, all critical computational time of used cryptographic primitives is given in Table 3.1 in the unit of milliseconds (ms), encompassing the average, maximum, and minimum runtimes. Since the given times are based on one voter and one election authority scenario, we exclude the cost of aggregate credentials in TIAC which does not affect the overall efficiency of the system. Using this table,

Table 3.1: Time consumption of used cryptographic primitives

Primitive	Algorithms	Notation	Average Time	Max Time	Min Time
Exponential El-Gamal	Encryption	$\mathsf{T}_{\mathsf{Enc}}$	39.40	40.07	39.22
	Partial Decryption	$\mathsf{T}_{\mathsf{ParDec}}$	59.97	60.08	59.24
Chaum-Pedersen	Prove Vote Validity	$\mathsf{T}_{\mathsf{PrVV}}$	129.85	130.26	127.91
	Verify Vote Validity	$\mathsf{T}_{\mathsf{VrVV}}$	179.81	181.30	158.17
	Prove Vote Sum	$\mathsf{T}_{\mathsf{PrVS}}$	39.92	40.81	38.30
	Verify Vote Sum	$\mathsf{T}_{\mathsf{VrVS}}$	99.74	103.72	85.64
	Prove Correct Decryption	$\mathsf{T}_{\mathsf{PrCD}}$	60.18	61.19	59.07
	Verify Correct Decryption	$\mathsf{T}_{\mathsf{VrCD}}$	80.07	83.27	78.48
TIAC	Prepare Blind Signature	$\mathsf{T}_{\mathsf{PrepBSign}}$	3.57	4.69	3.12
	Blind Signature	$\mathsf{T}_{\mathsf{BSign}}$	4.12	4.89	3.13
	Unblind Signature	$\mathsf{T}_{\mathsf{UnBlind}}$	26.89	28.57	25.41
	Prove Credential	$\mathsf{T}_{\mathsf{PrCred}}$	3.69	4.07	3.16
	Verify Credential	$\mathsf{T}_{\mathsf{VrCred}}$	24.74	26.64	23.18

we can analyze each stage of an election, and compare the performance of Proba with Blockchain-powered Helios.

In the pre-election stage, while Helios sends an email to its voters with their assigned credentials to be used in the election, Blockchain-powered Helios authorizes voters' chosen wallet addresses as credentials by uploading them to the smart contract. On the contrary, Proba authorizes voters' chosen wallet public keys that are bound to wallet addresses through a cryptographic primitive known as $\mathcal{TIAC}$. Thus, we can just give the computational time cost of the used cryptographic primitive in the pre-election stage of Proba as $\mathsf{T}_{\mathsf{pre}}$. Let n_v denote the number of voters, and t is the threshold of election authorities. Based on Table 3.1, pre-election time cost can be computed as follows:

$$\begin{aligned}\mathsf{T}_{\mathsf{pre}} &= n_v \cdot (t \cdot (\mathsf{T}_{\mathsf{PrepBSign}} + \mathsf{T}_{\mathsf{BSign}} + \mathsf{T}_{\mathsf{Unblind}}) + \mathsf{T}_{\mathsf{PrCred}}) \\ &= n_v \cdot (t \cdot 34.58 + 3.69)\text{ms}\end{aligned}$$

In the election stage, voters in Helios send their encrypted votes and ZKPs with their credentials to the server, then take the hash of the encrypted vote as the ballot tracker. Voters in Blockchain-powered Helios send their encrypted votes and ZKPs to IPFS, and take its IPFS address (digest of the file) to initiate a transaction, then take the hash of the signed transaction which includes the IPFS address. In this case, the cost of IPFS (T_{IPFS}) is added to the system. In the election phase of Proba, voters send their encrypted votes, ZKPs, and proof of randomized anonymous credentials to IPFS. This stored credential in IPFS will allow voters to check whether each voter is eligible or not. After taking the IPFS address, voters initiate a transaction that includes the IPFS

link and the proof of their randomized anonymous credentials, then take the hash of the signed transaction. Let n_c denote the number of candidates, the time spent during the election phase $\mathsf{T}_{\mathsf{elec}}$ without IPFS cost T_{IPFS} in Proba can be given as:

$$\begin{aligned}\mathsf{T}_{\mathsf{elec}} &= n_v \cdot (n_c \cdot (\mathsf{T}_{\mathsf{Enc}} + \mathsf{T}_{\mathsf{PrVV}}) + \mathsf{T}_{\mathsf{PrVS}} + \mathsf{T}_{\mathsf{PrCred}} + \mathsf{T}_{\mathsf{VrCred}}) \\ &= n_v \cdot (n_c \cdot 169.25 + 68.35)\text{ms}\end{aligned}$$

In the post-election stage, each system uses the same additively homomorphic encryption and ZKPs as a cryptographic primitive, thus this stage doesn't affect the efficiency of Proba. The time spent of post-election $\mathsf{T}_{\mathsf{post}}$ in Proba can be defined as:

$$\begin{aligned}\mathsf{T}_{\mathsf{post}} &= n_v \cdot (n_c \cdot \mathsf{T}_{\mathsf{VrVV}} + \mathsf{T}_{\mathsf{VrVS}}) + t \cdot (\mathsf{T}_{\mathsf{ParDec}} + \mathsf{T}_{\mathsf{PrCD}} + \mathsf{T}_{\mathsf{VrCD}}) \\ &= n_v \cdot (n_c \cdot 179.81 + 99.74) + t \cdot 161.53\text{ms}\end{aligned}$$

In conclusion, Proba has two key stages that affect its efficiency, including authorizing the voter's wallet public key as a credential (pre-election) and sending vote transactions with the voter's credential proof (election). Since the original Blockchain-powered Helios do not solve the credential authorization problem with a cryptographic primitive during pre-election, we focus on the efficiency comparison of the election stage. It should be noted that Blockchain-powered Helios doesn't have the cost of $\mathsf{T}_{\mathsf{PrCred}}$ and $\mathsf{T}_{\mathsf{VrCred}}$ in the election stage since it authorizes the credential by storing. Figure 3.15 shows time spent in the election stages of Proba and Blockchain-powered Helios in terms of second (s).

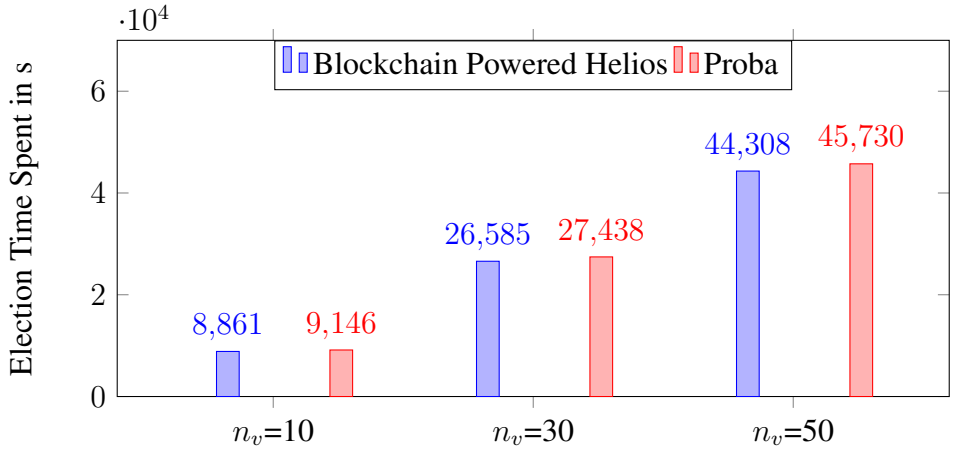

Figure 3.15: Time cost (s) comparison of election stage when $t = 3$, $n_c = 5$

On the other hand, we can also compare the efficiency in terms of smart contract storage cost. It can be seen that the usage of $\mathcal{TIAC}$ has a positive impact on the smart contract cost. The storage operation in a smart contract (20.000 Gas unit per 256 bit) is the most consuming operation after creating and calling an account operation [149], thus this operation needs to be as little as possible. While Blockchain-powered Helios stores each voter's wallet addresses in smart contracts, Proba doesn't require to store wallet address beforehand due to $\mathcal{TIAC}$. Thus, the storage execution cost in Blockchain-powered Helios is saved in Proba. Considering the Ethereum wallet address is 160 bit, Figure 3.16 shows the storage cost of Proba and Blockchain-powered Helios in the Gas unit. The lower storage cost of the smart contract makes the Proba system more accessible and cost-effective. Also, it helps to improve the scalability of a blockchain by limiting the amount of data in on-chain storage.

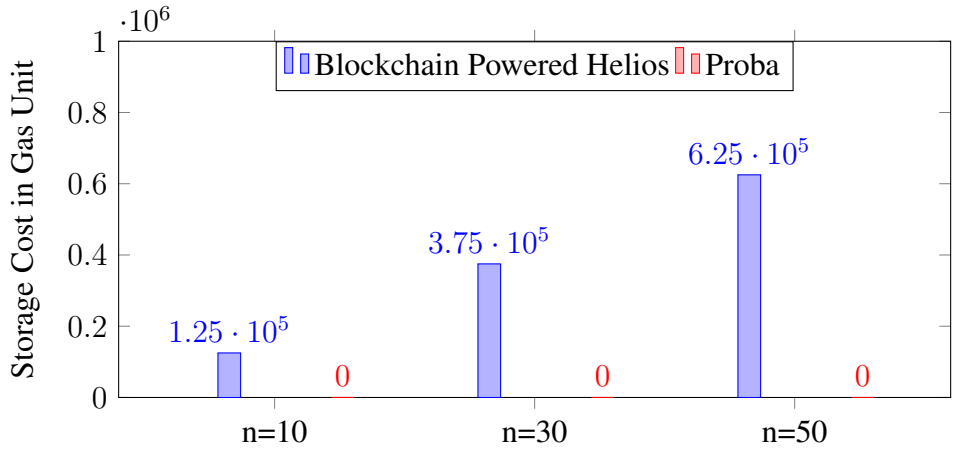

Figure 3.16: Smart contract storage cost where n is the number of the voter

Remark 4. Since Proba utilizes the Hyperledger Fabric as its consortium blockchain, it does not incur any gas costs, in contrast to Ethereum where each transaction necessitates a certain amount of gas. However, if the Proba system were to be executed on the Ethereum blockchain, each individual transaction would incur a gas cost amounting to 21.000 gas units. Furthermore, the storage of n-aggregated votes would necessitate a gas cost of $20.000 + (n-1) \cdot 5.000$ units. Moreover, the verification of the vote's validity and the voter's credential necessitates the execution of implemented Ethereum Precompiled contracts on the AltBN128 curve [29]. These contracts are designed to define the necessary operations, such as elliptic curve addition (150 gas unit), elliptic curve multiplication (6.000 gas unit), and pairing checks ($34.000 \cdot k + 45.000$ gas unit where k is the number of pairing).

CHAPTER 4

WALLET KEY PROTECTION PROTOCOLS

This chapter presents efficient threshold signature protocols to protect the wallet secret key by splitting it across different devices of the same voter in general blockchain-based I-voting systems. These protocols can be analyzed in two different sections. The first section introduces an online-friendly (single-signature verification cost) two-party ECDSA protocol against malicious adversaries with reduced computation and communication costs for the offline phase. The second section presents a flexible hierarchical threshold signature scheme, referred to as FlexHi, which is constructed based on Shamir's method but incorporates independent polynomials at each hierarchical level.

4.1 EFFICIENT SECURE TWO-PARTY ECDSA FOR KEY PROTECTION

This section introduces a new two-party signing protocol on the Elliptic Curve Digital Signature Algorithm (ECDSA) for the protection of a voter's signing key. ECDSA [90] is the elliptic curve variant of the digital signature algorithm [73]. In the two-party version of ECDSA, two parties execute the key generation phase and signing phase as an interactive protocol, while the verification phase remains the same as in the classical signature [140].

In blockchain content, an efficient two-party ECDSA scheme is golden for protection against cryptocurrency theft. Protecting signing keys in ECDSA has the same meaning as protecting cryptocurrency. Thus, it is not logical to store the signing key in one place to protect it. Instead, the signing key must be shared with at least two places

in a blockchain-based I-voting system. The study in this thesis enhances the two-party ECDSA protocol by reducing the bandwidth, unlike the ECDSA techniques described in the existing literature. By utilizing this protocol, voters can securely use their signing key with their shared devices.

4.1.1 Building Blocks

Multi-party computation (MPC) is a technique from cryptography that enables multiple parties to conduct computation on their secrets while preserving them private. MPC was formally introduced with Yao's 2-party protocol for the Millionaires' problem [152]. Today, it became a pioneering solution for a wide variety of real-world problems, such as cryptographic key protection, privacy-preserving data analytics, and so forth [82].

With the rise of the blockchain technology and cryptocurrencies, multi-party signing [42] and, in particular, threshold signing has gained significant attention in the past decade. Namely, a (t, n) signature scheme enables n parties to distribute the signing power in such a way that signing a message m requires the collaboration of at least $t + 1$ of them. This is accomplished by having the n parties participate in the key generation phase to produce a private key unknown to them. At the end of this phase, each party will hold a share of the private key, together with the public key. Then the signing phase is executed as an interactive protocol as well, where at least $t + 1$ parties participate with their shares so as to produce the signature, which is then checked with the verification algorithm of the signature scheme being distributed. This benefits cryptocurrencies as transactions are sent by producing a signature using the sender's private key. Thus to prevent a single point of failure while maintaining the key, one can share it among different parties placed in different locations, who need to collaborate to sign.

4.1.1.1 ECDSA Scheme

The ECDSA scheme [90] is a signature algorithm that involves key generation, signing and verification. Let G be an elliptic curve group of order q with base point P. The

ECDSA scheme works as follows:

- **KeyGen**$(1^\lambda) \rightarrow (x, Q)$**:** sets a random private key $x \leftarrow Z_q$ and compute public key $Q = x \cdot P$
- **Sign**$(x, m) \rightarrow (r, s)$**:** generate the signature (r,s) using private key x and message m with hash function H
  - Set a random nonce $k \leftarrow Z_q$, compute $R = (r_x, r_y) = k \cdot P$, $r = r_x \mod q$
  - Compute $s = k^{-1}(H(m) + r \cdot x) \mod q$ and output (r, s)
- **Verification**$(m; (r, s)) \rightarrow (0, 1)$**:** gives 1 if the signature valid; 0 otherwise.
  - Compute $(r_x, r_y) = R = s^{-1} \cdot H(m) \cdot P + s^{-1} \cdot r \cdot Q$
  - If $r = r_x \mod q$, output 1; otherwise output 0.

Remark 1. In Elliptic curve mathematics if (r, s) is a valid signature, then its complement signature $(r, -s$ mod n) is also valid signature. Thus, it gives rise to the malleability problem of the ECDSA scheme. The best solution to overcome the malleability problem is the low-s rule where the low-s is the value between 0 and $\frac{q-1}{2}$. Therefore, we also accept the lower s value in the signature to be consistent.

Remark 2. In ECDSA, when the same random nonce k is employed in two signatures, then the private key can be recovered. This attack is referred to as a bad randomness attack [39] that was first reported in 2013 [120]. In detail, when $k_1 = k_2 = k$, then r value will be equal. In this case, $s_1 = k^{-1} \cdot (H(m_1) + r \cdot x)$ and $s_2 = k^{-1} \cdot (H(m_2) + r \cdot x)$, then attacker can find private key as follows:

$$x = \frac{H(m_1) \cdot s_2 - H(m_2) \cdot s_1}{r \cdot (s_1 - s_2)} \tag{4.1}$$

Thus for each signature, a new random nonce needs to be produced. To avoid this attack, Bitcoin Core has already implemented a novel function following RFC 6979 [109] that offers a deterministic nonce generation procedure depending on a message m and a private key x.

4.1.1.2 Threshold ECDSA Schemes

Thresholdizing the ECDSA algorithm has drawn most of the attention, as it is the signing algorithm used in Bitcoin. It can be found in the literature many works that addressed this [81, 52, 43, 151, 28, 44]. Those schemes differ particularly in the way of sharing values, namely additively or multiplicatively. That is, at the heart of the ECDSA algorithm, one needs to calculate $s = k^{-1}(H(m) + x \cdot r) \mod q$. In a threshold version of ECDSA, both the private key x and the random nonce k used for signing the message m are secretly shared among parties. In fact, to provide a threshold version of ECDSA, the main challenge consists of choosing an adequate way to secretly share k and x so that s can be computed efficiently. Note that this calculation contains inverting a secret, and multiplying it with another value obtained by evaluating linear operations over another secret (addition and multiplication with opened values).

For instance, for the 2-party case, additively secret sharing k is problematic for inversion, as in this case, party P_1 holds k_1 and party P_2 holds k_2 subject to $k_1 + k_2 = k \mod q$, and from this, the two parties need to calculate k^{-1}. One can alternatively secretly share k in a multiplicative way to overcome this obstacle, as in this case, inverting becomes a local operation; however, the resulting value still needs to be multiplied by $H(m) + x \cdot r$, which still induces obstacles either x was additively or multiplicatively secret shared.

As a solution to these challenges, several authors in the field proposed using homomorphic encryption. This approach allows one party to transmit a secret the other party in encrypted form so that they can execute the challenging computation and decrypt it afterward. The homomorphic encryption schemes that were used are partially homomorphic, as performing one type of operations over the ciphertexts was sufficient for the computation needed.

For the most part, homomorphic encryption was introduced to realize a Multiplicative-to-Additive (MtA) functionality which enables parties to obtain an additive version of the shares of a secret from a multiplicative one, adopting ideas from [86]. Therefore, parties can query this functionality when an additive sharing is preferable than

a multiplicative one from a performance point of view. Of course, this functionality does not come for free, and it induces a cost to the protocol whenever it is called; however, there exist many instantiations of it, such as Paillier-based MtA [52], ElGamal-based MtA [83], and CL-based MtA [30]. Besides, one can also construct MtA based on Oblivious transfer (OT) [43], which has the advantage of decreasing the computational complexity by eliminating the need for homomorphic encryption at the expense of incurring a relatively high bandwidth. As a result, one has multiple options for MtA instantiations, each of which offers a different tradeoff between the computation and communication costs, thanks to which one can select the one that best fits the constraints faced.

For the 2-party case of threshold ECDSA, two works are most related to the work in this thesis, namely, the one of Lindell [81] and Xue et al. [151]. Lindell has proposed a simple and efficient 2-party protocol against malicious adversaries. To briefly go over this protocol, both x and k are secretly shared in a multiplicative way, where each party P_i generates x_i in the key generation phase so that the private key x is equal to $x = x_1 \cdot x_2$. Party P_1 also encrypts x_1 so as to send it to P_2, then in the signing phase, the two parties generate their share of the nonce k, then P_2 computes its share of s and sends it to P_1, which involves encrypting and performing homomorphic encryption operations. Finally, P_1 calculates the signature s, which involves decryption before the verification step.

On the other hand, Xue et al. proposed an online-friendly algorithm against malicious adversaries. That is, this protocol has a nearly optimal online phase, in the sense that the heaviest part of it consists of the verification step of the signature, which in turn consists of calculating two scalar multiplications of elliptic curve points (scalar multiplications will be denoted as M from now on). The communication cost is also efficient, as only a single field element needs to be sent. This is opposed to [81] as one needs to send and operate over ciphertexts during the online phase. However, providing such an efficient online phase came with the cost of offloading all the heavy computation in the offline phase of the signing step. That is, while the key generation does not involve any encryption, an MtA is being executed for every signature during the signing phase, which is still a good compromise as it reduces the number of calls to the MtA functionality compared to other schemes. Thus the resulting protocol

offers an efficient online phase with a good overall cost. However, this scheme can be further optimized, as we will see in the next sections.

4.1.2 Ideal Functionality for Two-Party ECDSA

It is aimed to describe below the ideal $\mathcal{F}_{\mathsf{2ECDSA}}$ functionality that the protocol in this thesis realizes, as well as the ideal functionalities queried by the protocol in this thesis, namely, an ideal zero-knowledge proof functionality $\mathcal{F}_{\mathsf{ZKP}}$ and an ideal committed non-interactive zero-knowledge functionality $\mathcal{F}_{\mathsf{Commit-ZK}}$ which are similar to the ones used in [81], as well as an ideal Multiplicative-to-Additive (MtA) functionality $\mathcal{F}_{\mathsf{MtA}}$.

4.1.2.1 $\mathcal{F}_{\mathsf{2ECDSA}}$ Functionality

The $\mathcal{F}_{\mathsf{2ECDSA}}$ functionality is composed of a key generation phase and a signing phase. In the key generation phase, the key pair (x, Q) is generated, where x is stored internally, and Q is given to the parties. In the signing phase, the signature on the given message is constructed and given to $\mathcal{P}_1$. The functionality is introduced in Figure 4.1.

2-party ECDSA functionality $\mathcal{F}_{\mathsf{2ECDSA}}$

Given an elliptic curve group $\mathbb{G}$ of order q, a generator P of $\mathbb{G}$, and a hash function H with a codomain of size λ bits. The functionality works as follows:

KeyGen: On input init from both parties $\mathcal{P}_1$ and $\mathcal{P}_2$:

- Run KeyGen as defined in ECDSA scheme, so as to generate a key pair (x, Q).
- Store (x, Q) and send Q to both parties.
- Set an internal flag ready to 1 and ignore further calls.

Sign: On input $\mathsf{Sign}(sid, m)$ from both parties $\mathcal{P}_1$ and $\mathcal{P}_2$. If $\mathsf{ready} = 1$ and *sid* has not been used previously:

- Run Sign as defined in ECDSA scheme, so as to generate the signature (r, s).
- Send (r, s) to $\mathcal{P}_1$.
- Store internally $(sid, \mathsf{delivered})$.

Figure 4.1: 2-party ECDSA functionality $\mathcal{F}_{\mathsf{2ECDSA}}$

4.1.2.2 $\mathcal{F}_{\mathsf{ZKP}}$ Functionality

The $\mathcal{F}_{\mathsf{ZKP}}$ functionality is depicted in Figure 4.2. With this functionality, one party can prove the knowledge of a witness w for an element x, such that the pair (x, w)

satisfies the relation $\mathcal{R}$. For the proposed protocol in this thesis, this relation is $\mathcal{R} \leftarrow \{(Q, x) \in \mathbb{G} \times Z_q | Q = [x] \cdot P\}$ for public parameters $\mathbb{G}$ and its generator P, which allows to prove knowledge of the discrete log of an elliptic curve point. Sigma protocol of Schnorr [121] can be used to instantiate this functionality, which can be made non-interactive using the Fiat-Shamir paradigm in the random-oracle model [48].

$\mathcal{F}_{\mathsf{ZKP}}$

$\mathcal{F}_{\mathsf{ZKP}}$ functionality between $\mathcal{P}_1$ and $\mathcal{P}_2$ works as follows:

Prove: On input $(prove, sid, x, w)$ from $\mathcal{P}_i$ for $i \in \{1, 2\}$, send $(proof, sid, x)$ to $\mathcal{P}_{3-i}$ if $(x, w) \in R$ and *sid* has not been previously used, otherwise ignore the message.

Figure 4.2: $\mathcal{F}_{\mathsf{ZKP}}$

4.1.2.3 $\mathcal{F}_{\mathsf{Commit-ZK}}$ Functionality

The $\mathcal{F}_{\mathsf{Commit-ZK}}$ functionality is depicted in Figure 4.3. Through this functionality, a party will be able to commit to its Non-interactive ZKP (NIZKP) and open it afterward. As mentioned in [81], this functionality can be realized in the random oracle model by having the parties hash their NIZKP concatenated with a randomness r, which will be both opened in the decommitment phase.

$\mathcal{F}_{\mathsf{Commit-ZK}}$

$\mathcal{F}_{\mathsf{Commit-ZK}}$ functionality between $\mathcal{P}_1$ and $\mathcal{P}_2$ works as follows:

Commit: On input $(\mathsf{com-prove}, sid, x, w)$ from $\mathcal{P}_i$ for $i \in \{1, 2\}$, record (sid, i, x, w) if *sid* has not been used previously and $(x, w) \in R$, then send $(\mathsf{proof-receipt}, sid)$ to $\mathcal{P}_{3-i}$, otherwise ignore the message.

Decommit: On input $(\mathsf{decom-proof}, sid)$ to $\mathcal{P}_i$, send $(\mathsf{decom-proof}, sid, x, 1)$ to $\mathcal{P}_{3-i}$ if (sid, i, x, w) is recorded and $(x, w) \in R$, otherwise send $(\mathsf{decom-proof}, sid, x, 0)$ to $\mathcal{P}_{3-i}$

Figure 4.3: $\mathcal{F}_{\mathsf{Commit-ZK}}$

4.1.2.4 $\mathcal{F}_{\mathsf{MtA}}$ Functionality

The $\mathcal{F}_{\mathsf{MtA}}$ functionality is depicted in Figure 4.4. This functionality takes as an input the two values α and β coming from $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and forwards to them respectively two random values a and b, subject to the relation $a + b = \alpha \cdot \beta \mod q$, i.e., it transforms a multiplicative sharing of a secret to an additive sharing. As stated earlier, one can instantiate MtA from many constructions, such as the Paillier encryption scheme [99] or El Gamal [46], class groups or OT.

$\mathcal{F}_{\mathsf{MtA}}$

$\mathcal{F}_{\mathsf{MtA}}$ functionality between $\mathcal{P}_1$ and $\mathcal{P}_2$ works as follows:

Reshare: On input $(sid, \alpha \in Z_q)$ from $\mathcal{P}_1$ and $(sid, \beta \in Z_q)$ from $\mathcal{P}_2$. If *sid* has been used before ignore this message. Otherwise:

- Sample $a \leftarrow Z_q$ and calculate $b \leftarrow \alpha \cdot \beta - a \bmod q$
- Send (sid, a) to $\mathcal{P}_1$ and (sid, a) to $\mathcal{P}_2$.

Figure 4.4: $\mathcal{F}_{\mathsf{MtA}}$

4.1.3 The Proposed Efficient Two-party ECDSA Protocol

The proposed two party ECDSA protocol is composed of two phases; one phase for a distributed key generation that runs once, at the end of which the parties will hold an additive sharing of the secret x as $x = x_1 + x_2$, then the second phase is for signing, which consists of:

- Generating the nonce k, which will be multiplicatively shared between the parties as $k = k_1 \cdot k_2$.

- Querying the MtA functionality, so as to convert the product of P_1's secret key x_1 and P_2's nonce k_2^{-1} to an additive sharing $a + b$, namely, P_1 and P_2 receive a and b respectively such that $a + b = x_1.k_2^{-1} \mod q$. After the query, $\mathcal{P}_1$ computes $Z \leftarrow [a] \cdot P$ and sends it to $\mathcal{P}_2$, who computes $(Z + [b] \cdot P) \cdot k_2$ and checks if it is equal to Q_1, so as to control the correctness of the MtA input against a malicious $\mathcal{P}_1$.

- Online signing, that starts by P_2 generating locally its share of the signature after the MtA invocation, namely $s_2 = k_2^{-1}(H(m) + r.x_2) + b \cdot r \mod q$, then sends it to P_1 who will generate the signature by calculating locally $s = k_1^{-1}(s_2 + a \cdot r) \mod q$ and verifying whether this signature is valid. Note that the nonce generation and the MtA invocation are message-independent, thus we can refer to these two steps as the offline signing.

The complete process is illustrated in Figure 4.5. Also the graphical representation of the key distribution and signing phase are given in Figure 4.6 and Figure 4.7, respectively.

2-party ECDSA Protocol

Given an elliptic curve group $\mathbb{G}$ of order q and a generator P of $\mathbb{G}$:

Key Generation: To generate a pair of keys for the ECDSA algorithm, the parties do as follows:

1. $\mathcal{P}_1$ generates $x_1 \leftarrow \mathbb{Z}_q$ and calculates $Q_1 = [x_1] \cdot P$.
2. $\mathcal{P}_1$ sends ($\mathsf{com-prove}$, 1, Q_1, x_1) to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
3. $\mathcal{P}_2$ receives ($\mathsf{proof-receipt}$, 1)
4. $\mathcal{P}_2$ generates $x_2 \leftarrow \mathbb{Z}_q$ and calculates $Q_2 = [x_2] \cdot P$.
5. $\mathcal{P}_2$ sends (prove, 2, Q_2, x_2) to $\mathcal{F}_{\mathsf{ZKP}}$.
6. $\mathcal{P}_1$ receives (proof,2, Q_2) from $\mathcal{F}_{\mathsf{ZKP}}$. If not, $\mathcal{P}_1$ aborts.
7. $\mathcal{P}_1$ sends ($\mathsf{decom-prove}$, 1) to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
8. $\mathcal{P}_2$ receives ($\mathsf{decom-proof}$, 1, Q_1, z) from $\mathcal{F}_{\mathsf{Commit-ZK}}$. If $z = 0$, $\mathcal{P}_2$ aborts.

Both parties set $Q \leftarrow Q_1 + Q_2$ to be the public key. The private key is $x \leftarrow x_1 + x_2 \mod q$ (note that no party holds x, but only an additive share of it).

Signing: To sign a message m, the parties do as follows:

1. **Generating the nonce k:**
   (a) $\mathcal{P}_1$ generates $k_1 \leftarrow \mathbb{Z}_q$ and calculates $R_1 = [k_1] \cdot P$.
   (b) $\mathcal{P}_1$ sends ($\mathsf{com-prove}$, $sid||1$ R_1, k_1) to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   (c) $\mathcal{P}_2$ receives ($\mathsf{proof-receipt}$, $sid||1$, 1)
   (d) $\mathcal{P}_2$ generates $k_2 \leftarrow \mathbb{Z}_q$ and calculates $R_2 = [k_2] \cdot P$.
   (e) $\mathcal{P}_2$ sends (prove, $sid||2$, R_2, k_2) to $\mathcal{F}_{\mathsf{ZKP}}$
   (f) $\mathcal{P}_1$ receives (proof, $sid||2$, R_2) from $\mathcal{F}_{\mathsf{ZKP}}$. If not, $\mathcal{P}_1$ aborts.
   (g) $\mathcal{P}_1$ sends ($\mathsf{decom-prove}$, $sid||1$) to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   (h) $\mathcal{P}_2$ receives ($\mathsf{decom-proof}$, $sid||1$, Q_1, z) from $\mathcal{F}_{\mathsf{Commit-ZK}}$. If $z = 0$, $\mathcal{P}_2$ aborts.

   Both parties set $R \leftarrow [k_1 \cdot k_2] \cdot P = (r, y)$, corresponding to the nonce $k \leftarrow k_1 \cdot k_2$ (note that no party holds k, but only a multiplicative share of it).
2. **Querying the MtA functionality:**
   (a) $\mathcal{P}_1$ and $\mathcal{P}_2$ query $\mathcal{F}_{\mathsf{MtA}}$ with the respective inputs x_1 and k_2^{-1}. $\mathcal{F}_{\mathsf{MtA}}$ forwards a to $\mathcal{P}_1$ and b to $\mathcal{P}_2$.
   (b) $\mathcal{P}_1$ calculates $Z \leftarrow [a] \cdot P$ and sends it to $\mathcal{P}_2$.
   (c) $\mathcal{P}_2$ verifies if $k_2 \cdot (Z + [b] \cdot P) = Q_1$. If it is not the case $\mathcal{P}_2$ aborts.
3. **Online signing:**
   - $\mathcal{P}_2$ calculates $s_2 \leftarrow k_2^{-1} \cdot (H(m) + x_2 \cdot r) + b \cdot r \mod q$ and sends it to $\mathcal{P}_1$.
   - $\mathcal{P}_1$ calculates $s \leftarrow k_1^{-1} \cdot (s_2 + a \cdot r) \mod q$.
   - $\mathcal{P}_1$ verifies if s is a valid signature of m, if so $\mathcal{P}_1$ outputs (r, s) as the signature.

Figure 4.5: 2-party ECDSA Protocol

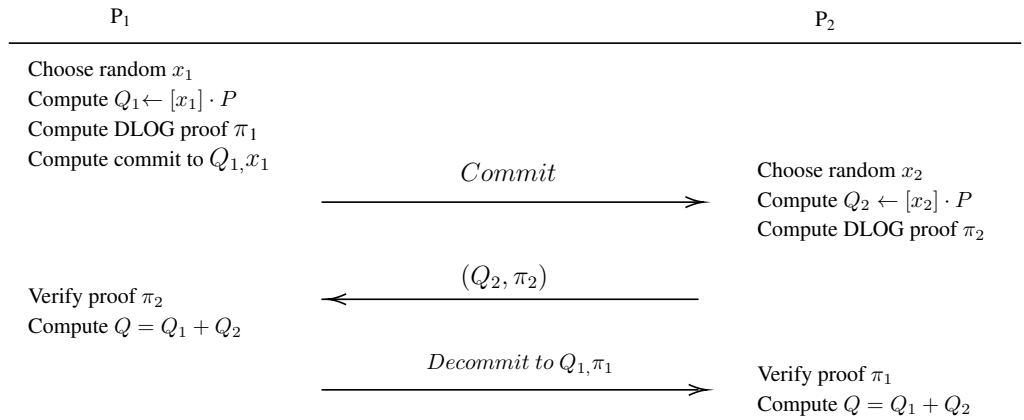

Figure 4.6: The 2-Party ECDSA Key Distribution Protocol

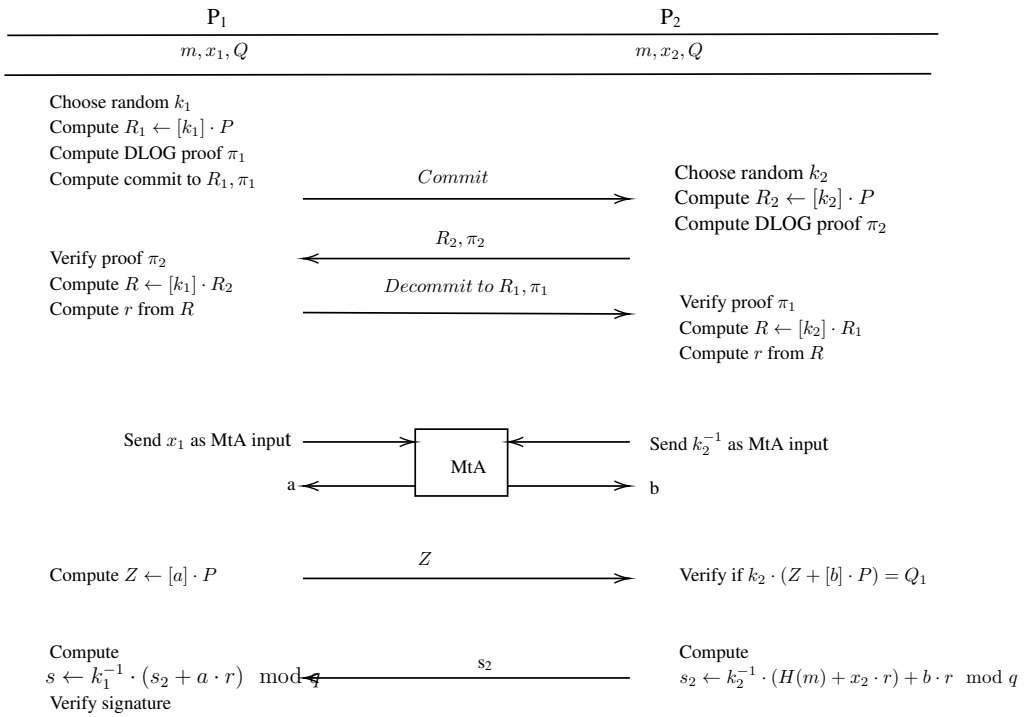

Figure 4.7: The 2-Party ECDSA Signing Protocol

Remark 3. The underlying concept of the protocol is to utilize the secret key x_1 belonging to party $\mathcal{P}_1$ and the secret nonce k_2 belonging to the other party $\mathcal{P}_2$ in the context of the MtA functionality. At the end of this process, $\mathcal{P}_2$ generates its partial signature, and then $\mathcal{P}_1$ adds its secrets to generate the full signature. It is important to note that, due to the absence of symmetry, this protocol cannot be directly applied to scenarios involving multiple parties. To address this constraint, an n-party system needs to be redesigned with different inputs of the MtA.

4.1.4 System Analysis

4.1.4.1 Security Analysis

Security of the proposed protocol is simulation-based, following the real / ideal paradigm [148]. The type of adversary we considered is a malicious one with static corruption. This implies that the adversary $\mathcal{A}$ can deviate from the protocol, but the party he corrupts (either P_1 or P_2) is set prior to the protocol execution. The proof of security is given in the following section.

Theorem. The protocol of Figure 4.5 securely implements the functionality of Fig-

ure 4.1 in the ($\mathcal{F}_{\mathsf{ZKP}}, \mathcal{F}_{\mathsf{Commit-ZK}}, \mathcal{F}_{\mathsf{MtA}}$)-hybrid model in the presence of a malicious static adversary under the idea/real definition of [148], assuming the Computational Diffie-Hellman problem is hard.

In Figure 4.8, a simulator $\mathcal{S}$ is built to simulate $\mathcal{P}_1$ when $\mathcal{P}_2$ is corrupt, and to simulate $\mathcal{P}_2$ when $\mathcal{P}_1$ is corrupt. Below, a sketch of proof demonstrates why the views in a real and a simulated execution will be indistinguishable for an adversary $\mathcal{A}$.

a.Corrupted P_1 Case

Key generation phase. The difference between the real execution and the simulated execution is the generation of Q_2. In the case of a real execution, Q_2 is computed as $[x_2] \cdot P$ where x_2 is randomly generated, while in the case of a simulated run, Q_2 is computed by calculating $Q_2 \leftarrow Q - Q_1$. Since Q is randomly generated by the $\mathcal{F}_{\mathsf{2ECDSA}}$ functionality of Figure 4.1 ($Q \leftarrow [x] \cdot P$ for a randomly generated x), then the distributions from which Q_2 is generated in the real and simulated executions are indistinguishable.

Signing phase. In the nonce generation, a similar argument can be given to show that the views are indistinguishable. That is in a real execution, R_2 is computed as $[k_2] \cdot P$ where k_2 is randomly generated, while in the case of a simulated run, R_2 is computed by calculating $R_2 \leftarrow [k_1^{-1}] \cdot R$. Since R is randomly generated by $\mathcal{F}_{\mathsf{2ECDSA}}$ ($R \leftarrow [k] \cdot P$ for a randomly generated k), then the distributions from which R_2 is generated in the real and simulated executions are indistinguishable.

In the MtA call, both in real and simulated executions, $\mathcal{P}_1$ is intended to receive a randomly generated a, thus the views are indistinguishable. Afterwards, $\mathcal{P}_1$ sends Z to $\mathcal{P}_2$. In a simulated execution, $\mathcal{P}_2$ aborts if $\mathcal{P}_1$ has provided to the MtA functionality a different input than x_1, or if he sends a different value than $[a] \cdot P$. This behavior is equivalent to what happens in a real execution, where $\mathcal{P}_2$ checks whether $k_2 \cdot (Z + [b] \cdot P) = Q_1$. That is, let us denote by ϵ_1, the additive error that $\mathcal{P}_1$ can introduce to x, namely, $\mathcal{P}_1$ sends to MtA the value $x' \leftarrow x + \epsilon_1 \mod q$, and by $[\epsilon_2] \cdot P$, the additive error that $\mathcal{P}_1$ can introduce to Z, namely, $\mathcal{P}_1$ sends $\mathcal{P}_2$ the value $Z' \leftarrow Z + [\epsilon_2] \cdot P$. To pass the check of $\mathcal{P}_2$, the following equation needs to be satisfied:

2-party ECDSA Simulator

The simulator $\mathcal{S}$ does as follows:

Corrupt $\mathcal{P}_1$ (i.e. simulating $\mathcal{P}_2$):

1. **Key Generation:**
   - $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{2ECDSA}}$ to obtain the public key Q.
   - $\mathcal{S}$ receives ($\mathsf{com-prove}$, 1, Q_1, x_1) from $\mathcal{A}$ intended to be sent to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   - $\mathcal{S}$ checks whether $Q_1 = [x_1] \cdot P$, if it is the case, $\mathcal{S}$ calculates $Q_2 = Q - Q_1$, and sends (proof,2 Q_2) to $\mathcal{A}$, as if $\mathcal{F}_{\mathsf{ZKP}}$ sent it. If Q_1 is different than $[x_1] \cdot P$, $\mathcal{S}$ does the same with a randomly generated Q_2.
   - $\mathcal{S}$ receives ($\mathsf{decom-proof}$, 1, Q_1, z) from $\mathcal{F}_{\mathsf{Commit-ZK}}$. If $z = 1$, the simulator stores (x_1, Q) for further use, otherwise, $\mathcal{S}$ simulates $\mathcal{P}_2$ aborting.
2. **Signing:**
   (a) **Nonce generation:**
   - $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{2ECDSA}}$ to obtain the signature (r, s), then calculates $R \leftarrow [s^{-1} \cdot H(m)] \cdot P + [s^{-1} \cdot r] \cdot Q$ as in the verification procedure.
   - $\mathcal{S}$ receives ($\mathsf{com-prove}$, $sid||1$, R_1, k_1) from $\mathcal{A}$ intended to be sent to $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   - $\mathcal{S}$ checks whether $R_1 = [k_1] \cdot P$, if it is the case, $\mathcal{S}$ calculates $R_2 = k_1^{-1} \cdot R$, and sends (proof, $sid||2$, R_2) to $\mathcal{A}$, as if $\mathcal{F}_{\mathsf{ZKP}}$ sent it. If R_1 is different than $[k_1] \cdot P$, $\mathcal{S}$ does the same with a randomly generated R_2.
   - $\mathcal{S}$ receives ($\mathsf{decom-proof}$, 1, R_1, z) from $\mathcal{F}_{\mathsf{Commit-ZK}}$. If $z = 1$, the simulator stores (k_1, R) for further use, otherwise, $\mathcal{S}$ simulates $\mathcal{P}_2$ aborting.

   (b) **MtA**:
   - The simulator here receives x_1 from $\mathcal{A}$ intended to be sent to $\mathcal{F}_{\mathsf{MtA}}$, then forwards a randomly generated number a to $\mathcal{P}_1$. If the x_1 received here is different from the share of the secret key of $\mathcal{P}_1$, the simulator sets an internal flag $\mathsf{cheat}_{sid||1}$ to be one.

   (c) **Online signing**:
   - If $\mathsf{cheat}_{sid||1}$ is equal to 1, $\mathcal{S}$ generates a random $s_2 \leftarrow Z_q$ and sends it to $\mathcal{P}_1$. If not, $\mathcal{S}$ calculates $s_2 \leftarrow s \cdot k_1 - a \cdot r$ and sends it to $\mathcal{P}_1$.

Corrupt $\mathcal{P}_2$ (i.e. simulating $\mathcal{P}_1$):

1. **Key Generation:**
   - $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{2ECDSA}}$ to obtain the public key Q.
   - $\mathcal{S}$ sends ($\mathsf{receipt}$, 1) to $\mathcal{A}$ as if it was sent by $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   - $\mathcal{S}$ receives (prove, 2, Q_2 , x_2) from $\mathcal{P}_2$ intended to be sent to $\mathcal{F}_{\mathsf{ZKP}}$.
   - $\mathcal{S}$ checks if $Q_2 = [x_2] \cdot P$. If it is not the case, $\mathcal{S}$ simulates $\mathcal{P}_1$ aborting.
   - $\mathcal{S}$ calculates $Q_1 = Q - Q_2$, and sends ($\mathsf{decom-proof}$, 1, Q_1, 1) as if $\mathcal{F}_{\mathsf{Commit-ZK}}$ sent it. $\mathcal{S}$ stores (x_2, Q) for further use.
2. **Signing:**
   (a) **Nonce generation:**
   - $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{2ECDSA}}$ to obtain the signature (r, s), then calculates $R \leftarrow [s^{-1} \cdot H(m)] \cdot P + [s^{-1} \cdot r] \cdot Q$ as in the verification procedure.
   - $\mathcal{S}$ sends ($\mathsf{receipt}$, $sid||1$, 1) to $\mathcal{A}$ as if it was sent by $\mathcal{F}_{\mathsf{Commit-ZK}}$.
   - $\mathcal{S}$ receives (prove, $sid||2$, R_2 , k_2) from $\mathcal{P}_2$ intended to be sent to $\mathcal{F}_{\mathsf{ZKP}}$.
   - $\mathcal{S}$ checks if $R_2 = [k_2] \cdot P$. If it is not the case $\mathcal{S}$ simulates $\mathcal{P}_1$ aborting.
   - $\mathcal{S}$ calculates $R_1 = k_2^{-1} \cdot R$, and sends ($\mathsf{decom-proof}$, 1, R_1, 1) as if $\mathcal{F}_{\mathsf{Commit-ZK}}$ sent it. $\mathcal{S}$ stores (k_2, R) for further use.

   (b) **MtA:**
   - The simulator here receives k_2^{-1} from $\mathcal{A}$ intended to be sent to $\mathcal{F}_{\mathsf{MtA}}$, then forwards a randomly generated number b to $\mathcal{P}_2$. If the k_2^{-1} received here is different from the one stored in the nonce generation, the simulator sets an internal flag $\mathsf{cheat}_{sid||2}$ to be one.

   (c) **Online signing:**
   - $\mathcal{S}$ receives s_2 from $\mathcal{A}$. If s_2 is different from $k_2^{-1} \cdot (H(m) + x_2 \cdot r) + b \cdot r$ or $\mathsf{cheat}_{sid||2} = 1$, $\mathcal{S}$ simulates $\mathcal{P}_1$ aborting. Otherwise, $\mathcal{S}$ outputs (r, s) as the signature.

Figure 4.8: 2-party ECDSA Simulator

$$
\begin{aligned}
Q_1 &= k_2 \cdot (Z' + [b] \cdot P) \\
&= k_2 \cdot (Z + [\epsilon_2] \cdot P + [b] \cdot P) \\
&= k_2 \cdot ([\epsilon_2] \cdot P + [a] \cdot P + [b] \cdot P) \\
&= k_2 \cdot ([\epsilon_2] \cdot P + (x_1 + \epsilon_1) \cdot k_2^{-1} \cdot P) \\
&= k_2 \cdot ([\epsilon_2] \cdot P + x_1 \cdot k_2^{-1} \cdot P + \epsilon_1 \cdot k_2^{-1} \cdot P) \\
&= Q_1 + k_2 \cdot ([\epsilon_2] \cdot P + \epsilon_1 \cdot k_2^{-1} \cdot P)
\end{aligned}
\tag{4.2}
$$

which implies that $k_2 \cdot \epsilon_2 + \epsilon_1 = 0 \mod q$. If $\epsilon_2 = 0$, then $\epsilon_1 = 0$ mod q. Also if $\epsilon_1 = 0$ mod q , then $\epsilon_2 = 0$ as $k_2 \neq 0$ mod q. Thus $\epsilon_2 = 0$ or $\epsilon_1 = 0$ mod q implies that the adversary has not cheated, as we end up with a case where he does not modify the values he is supposed to send. Let us look at the case where $\epsilon_2 \neq 0$ and $\epsilon_1 \neq 0$ mod q. The equation holds if the adversary chooses ϵ_1 in such a way that $\epsilon_2 = \epsilon_1 \cdot [k_2^{-1}] \cdot P = 0$. While $R_2 = [k_2] \cdot P$ is known to the adversary, obtaining $[k_2^{-1}] \cdot P$ from it would mean breaking the 1-Weak Diffie-Hellman problem, which is equivalent to the computational Diffie-Hellman problem as stated in [92].

Thus, to summarize, the adversary will not be able to make the check pass if he cheats, either in the MtA call or the step afterward.

In the online signing: If the parties reach this stage, $\mathcal{P}_1$ will be receiving in the simulated execution $s_2 = s \cdot k_1 - a \cdot r \mod q$, which is equal to

$$
\begin{aligned}
s_2 &= s \cdot k_1 - a \cdot r \\
&= k^{-1} \cdot (H(m) + r \cdot x) \cdot k_1 - a \cdot r \\
&= k_2^{-1} \cdot (H(m) + r \cdot x) - a \cdot r) \\
&= k_2^{-1} \cdot (H(m) + r \cdot x_1 + r \cdot x_2) - a \cdot r \\
&= k_2^{-1} \cdot (H(m) + r \cdot x_2) + k_2^{-1} \cdot r \cdot x_1 - a \cdot r \\
&= k_2^{-1} \cdot (H(m) + r \cdot x_2) + r \cdot (a + b) - a \cdot r \\
&= k_2^{-1} \cdot (H(m) + r \cdot x_2) + r \cdot b
\end{aligned}
\tag{4.3}
$$

which is what $\mathcal{P}_1$ receives in a real execution.

b.Corrupted P_2 Case

Key generation phase. Similarly to the case of a corrupted P_1, the difference between the real execution and the simulated execution is the generation of Q_1. In the case of a real execution, Q_1 is computed as $[x_1] \cdot P$ where x_1 is randomly generated, while in the case of a simulated run, Q_1 is computed by calculating $Q_1 \leftarrow Q - Q_2$. Since Q is randomly generated by the $\mathcal{F}_{\mathsf{2ECDSA}}$ functionality of Figure 4.1 ($Q \leftarrow [x] \cdot P$ for a randomly generated x), then the distributions from which Q_1 is generated in the real and simulated executions are indistinguishable.

Signing phase. Similarly to the case of a corrupted P_1, in the nonce generation, a similar argument can be given to show that the views are indistinguishable. That is in a real execution, R_1 is computed as $[k_1] \cdot P$ where k_1 is randomly generated, while in the case of a simulated run, R_1 is computed by calculating $R_1 \leftarrow [k_2^{-1}] \cdot R$. Since R is randomly generated by $\mathcal{F}_{\mathsf{2ECDSA}}$ ($R \leftarrow [k] \cdot P$ for a randomly generated k), then the distributions from which R_1 is generated in the real and simulated executions are indistinguishable.

In the MtA call, both in real and simulated executions, $\mathcal{P}_2$ is intended to receive a randomly generated b (in the simulated execution $b = x_1 \cdot k_2^{-1} - a \mod q$ for a randomly generated a), thus the views are indistinguishable. In the step afterward, in the simulated execution, P_1 receives $[k_2^{-1}] \cdot Q_1 - [b] \cdot P$, which is the same as what he receives in a real execution, as $[k_2^{-1}] \cdot Q_1 - [b] \cdot P = [k_2^{-1} \cdot x_1] \cdot P - [b] \cdot P = [a] \cdot P$. Thus the views are indistinguishable.

In the online signing:

- if $\mathcal{P}_2$ does not cheat at all during the protocol, he will be able to calculate $s_2 = k_2^{-1} \cdot (H(m) + x_2 \cdot r) + b \cdot r \mod q$. and send it to $\mathcal{P}_1$. In the real execution $\mathcal{P}_1$ will add it to its share s_1, and the sum will yield a valid signature which will be published by $\mathcal{P}_1$. In the simulated execution, s_2 will pass the check of the simulator and therefore he will publish the signature.

- if $\mathcal{P}_2$ cheated at the MtA call, or does not send the correct s_2, in the real execution, $\mathcal{P}_1$ will not find a valid signature after summing up its share with the one of $\mathcal{P}_2$, thus $\mathcal{P}_1$ will send the abort signal. In the simulated execution, either

cheat flag will be equal to 1 at this stage, or s_2 will not pass the check of the simulator. In both cases, the simulator will abort. That is, the only case where the views will be distinguishable, is when the adversary cheats on the MtA call, and yet manages to send the correct s_2. Let us denote by ϵ, the additive error that the adversary introduces to his input to MtA, namely he sends $k_2^{-1} + \epsilon$ instead of k_2^{-1}. In this case $a + b = x_1 \cdot (k_2^{-1} + \epsilon)$. In order to pass the check, $\mathcal{P}_2$ needs to send s_2 such that $s \cdot k_1 = s_2 + a \cdot r \mod q$. This implies that:

$$\begin{aligned} s_2 &= s \cdot k_1 - a \cdot r \\ &= k^{-1} \cdot (H(m) + r \cdot x) \cdot k_1 - a \cdot r \\ &= k_2^{-1} \cdot (H(m) + r \cdot x_1 + r \cdot x_2) - a \cdot r \\ &= k_2^{-1} \cdot (H(m) + r \cdot x_2) + k_2^{-1} \cdot r \cdot x_1 - a \cdot r \\ &= k_2^{-1} \cdot (H(m) + r \cdot x_2) + r \cdot b - x_1 \cdot r \cdot \epsilon \end{aligned} \tag{4.4}$$

As x_1 is unknown to the adversary, he can satisfy this equation only if $\epsilon = 0$, i.e., the case where he does not cheat in the MtA call. Thus the behaviour of the simulator will make the real execution and the simulated one indistinguishable.

4.1.4.2 Performance Analysis

The theoretical complexity of the proposed two-party ECDSA protocol is analyzed below and compared with the one of [151] and [81].

Theoretical complexity - key distribution. The distributed key generation consists of generating keys and zero-knowledge proofs. The computation cost can be examined in terms of EC multiplications as this is the heaviest operation performed. For the keys, each party carries out 1M to produce its share of the public key. On the other hand, two zero-knowledge proofs of knowledge of discrete log need to be produced. Using the standard Schnorr proofs in non-interactive from, each party carries out 1M as a prover and 2M as a verifier. Thus the key distribution requires 8M in total. For the communication cost, each party needs to send its share of the public key and the corresponding nizkp, and $\mathcal{P}_1$, needs to send as well a commitment to its share at the beginning of the protocol, which consists of an output of the hash function H being

used (of size λ bits). Assuming one EC point can be represented in λ bits, and a nizkp consists of two field elements and one EC point, the size of data communicated between the parties is $9 \cdot \lambda$. Note that the cost of the proposed protocol's key distribution is the same as [151], which is a negligible cost compared to the one of Lindell [81], as the latter is dominated by the usage of homomorphic encryption.

Theoretical complexity - signing. T The computation cost of the signing protocol can be examined in terms of EC multiplications and MtA invocations. That is, the first step of the offline phase is similar to the key generation, except that the nonce is multiplicatively shared, thus each party needs to perform an extra EC multiplication so as to obtain R. Also, the calculation needed to check $\mathcal{P}_1$'s input to MtA requires 3 EC multiplications. Thus, it results in a computation cost of 13M, and a communication cost of $10 \cdot \lambda$. To obtain the total cost of the offline phase, one needs to add these costs to the execution of 1 MtA. The cost of this depends on the instantiation used.

On the other hand, the online phase consists of performing operations over a field by both parties and a verification phase of the signature, which requires from the verifier (in our case $\mathcal{P}_1$) to carry out 2M. Thus neglecting the cost of operations over a field, the computation cost of the online phase if 2M. As for the communication cost, $\mathcal{P}_2$ needs to send one field element to $\mathcal{P}_1$, thus λ bits of data need to be communicated between the parties.

Table 4.1 compares these costs with the ones of [151] and [81]. For [81], the cost of the homomorphic operations is dominated by exponentiations modulo N^2 by numbers from Z_N. It is referred to these exponentiations as E. The value N refers to the public key of Paillier, which determines the size of a Paillier encryption, which is a number from Z_{N^2}. MtA refers to the cost of invoking an instantiation of the MtA functionality. As one can notice, the computation and communication cost of the proposed protocol's online phase is the same as [151], which outperforms the one of [81], for which the online phase requires performing extra exponentiation and sending an encryption of Paillier (N is typically of size 2048 bits) instead of a field element. However, the proposed protocol's offline phase outperforms the one of [151], as in our case the computation and communication required are reduced respectively by 1M, and $2 \cdot \lambda$.

Table 4.1: Cost Analysis of Signing

Protocol	Computation		Communication	
	Offline	Online	Offline	Online
Lindell [81]	10M+2E	2M+1E	$9 \cdot \lambda$	$2 \cdot log_2(N)$
Xue et al. [151]	14M+1MtA	2M	$12 \cdot \lambda$+1MtA	λ
Ours	13M+1MtA	2M	$10 \cdot \lambda$+1MtA	λ

Table 4.2: Runtimes in milliseconds of the proposed protocol

Key generation	Offline signing	Online signing
1.05	1.26 + MtA	0.10

Implementation. The proposed protocol is implemented in C++, over the secp256k1 curve standardized by NIST, which is the one used by Bitcoin. The used hash function is Sha256, and for the curve operation Secp256k1[1] C library is used. The implementation can be found in `https://github.com/YounesTal1/2ecdsa`

We took runtimes with an Amazon instance of "t2.xlarge" (16 GiB of memory and 4 vCPU), running with "Ubuntu 18.04.6 LTS", this instance was located in "us-east-1" (Virginia). Table 4.2 gives the obtained runtimes. These runtimes correspond to the time needed for one key generation, one execution of the offline phase, and one execution of the online phase. Note that this implementation uses a single thread and that the runtimes reflect only the computation cost of the proposed protocol. These runtimes were obtained by calculating the average time needed for a 1000-key generation, where each key was used to sign 100 messages. As can be observed, the proposed protocol is efficient in terms of the computation cost, for both key generation and signing. That is, the key generation only requires 1.05ms and the offline phase (excluding the MtA call) requires 1.26ms. The difference in runtimes is mainly due to the five extra EC multiplications, namely two extra EC multiplications that need to be performed for calculating R, as the nonce is shared multiplicatively, and three extra EC multiplications that need to be performed for checking the correctness of $\mathcal{P}_1$'s input to MtA. The online phase only requires 0.1ms, as this is dominated by two EC multiplications for signature verification.

[1] https://github.com/bitcoin-core/secp256k1

4.1.5 Summary

In this work, a protocol against malicious adversaries with a nearly optimal online phase as in [151] is proposed, but with reduced computation and communication costs for the offline phase. That is, the proposed protocol's key generation is the same as in [151], where additive secret sharings of x are produced, and the proposed protocol's online phase requires two M as in [151]. However, the proposed protocol's offline phase reduces the number of EC multiplications by one and the size of data communicated by two field elements.

The cost reduction is achieved by eliminating the additional step of re-sharing the secret x in [151] without compromising security. That is, at the heart of the signing phase of the protocol of [151], x was re-shared between the two parties (following obvious notation) as $x = x'_1 \cdot (k_2 + r_1) + x'_2$, where the nonce k is shared as $k = k_1 \cdot (r_1 + k_2)$, then the shares x'_1 and k_2 are the values forwarded to the MtA functionality.

Instead, the work in this thesis simplified the protocol by adopting a multiplicative sharing of k where it is unnecessary to perform a re-sharing step. This protocol query the MtA only once on the most convenient inputs for the choices. Namely, querying the MtA on x_1 as the input of $\mathcal{P}_1$, and k_2^{-1} as the input of $\mathcal{P}_2$. This was a logical choice as holding an additive sharing as

$$x_1 \cdot k_2^{-1} = a + b \mod q \tag{4.5}$$

by the players allows them to do the online phase in only one pass, as the signature s can be written as

$$s = k_1^{-1} \cdot (k_2^{-1} \cdot (H(m) + x_2 \cdot r) + x_1 \cdot k_2^{-1} \cdot r) \mod q \tag{4.6}$$

In this case, $\mathcal{P}_2$ computes locally its signature share as

$$s_2 \leftarrow k_2^{-1} \cdot (H(m) + x_2 \cdot r) + b \cdot r \mod q \tag{4.7}$$

and sends it to $\mathcal{P}_1$ to construct the signature

$$s \leftarrow k_1^{-1} \cdot (s_2 + a \cdot r) \mod q \tag{4.8}$$

However, it is crucial to note that the protocol requires $\mathcal{P}_1$ to input x_1 for MtA. If there are no checks on this input to MtA, a malicious $\mathcal{P}_1$ can corrupt the system since $\mathcal{P}_1$ takes the partial signature s_2 and then generates the full signature s. For example, a malicious $\mathcal{P}_1$ can forge a signature on a different message m' of his choice by crafting the value to be sent to MtA as $x_1' \leftarrow -r^{-1} \cdot (H(m') - H(m) + x_1 \cdot r)$, then $\mathcal{P}_1$ will compute the full signature s as $k_1^{-1} \cdot (s_2 + a \cdot r) = k_1^{-1} \cdot (k_2^{-1} \cdot (H(m) + x_2.r) + (a+b) \cdot r) = k^{-1} \cdot (H(m') + x \cdot r)$ which is a valid signature on m' that is chosen by $\mathcal{P}_1$. In brief, in order to prevent $\mathcal{P}_1$ from mounting such attacks and manipulating the distribution of s_2, we add a check operation on the correctness of the MtA input of $\mathcal{P}_1$. Namely after calling MtA and receiving its outputs, $\mathcal{P}_1$ computes $[a] \cdot P$ and sends it to $\mathcal{P}_2$, who computes $([a] \cdot P + [b] \cdot P) \cdot k_2$ and checks whether it is equal to Q_1 or not. The correctness of this equality ensures that $\mathcal{P}_1$ used the correct x_1 value as MtA input.

In sum, the proposed protocol utilized an additive sharing of x and a multiplicative sharing of k, which is a similar setting of [43] for the (2, n)-ECDSA case (i.e., any two parties among the n parties can construct a valid signature). However, the proposed protocol only call the MtA functionality once while it is being called three times in [43]. Besides, this protocol only perform 13M, while 16M are needed for [43].

This improvement has an impact depending on the instantiation of MtA. For instance, in the case of an OT-based MTA, where such a choice is usually made to have a low computation cost, reducing the number of EC multiplications by one will decrease the computation cost of the offline phase of [151] by 5.4 percent, assuming EC multiplication is the heaviest part of the protocol. On the other hand, in the case of a CL-based MtA, which induces a low communication cost, reducing the size of transmitted data by two field elements decreases the communication cost of the offline phase of [151] for the case of the secp256k1 curve by 3.7 percent.

4.2 FlexHi: A Flexible Hierarchical Threshold Signature Scheme For Key Protection

This section presents a flexible hierarchical threshold signature scheme (FlexHi) designed to protect the voter's wallet signing key by giving different levels of authorization to different devices. While threshold signatures have many applications, all participants have been assigned the same weight, but in real-life applications, hierarchy is necessary for many situations. Hierarchical threshold signatures extend the concept of threshold signatures.

In the e-voting system content, the hierarchy can be used on different signing devices of the same voter. The voter divides his signing key into multiple components while granting greater rights to specific components. During the account verification procedure of any application, smartphones become more important tool than other devices in confirming the identity of the person. Thus, the smartphone occupies the topmost position in the hierarchy. Considering this, the voter can split his wallet's signing key over his smartphone at the highest hierarchical level and his laptop and tablet at an equal hierarchical level. In the (2,3) scenario, the voter has the option to utilize one of his tablet or laptop throughout the signing process. Nevertheless, it is obligatory to utilize your phone for this process of signing.

The proposed FlexHi is a simple and efficient hierarchical threshold system based on Shamir's secret sharing [124]. However, it utilizes independent polynomials at each hierarchical level, enabling us to accommodate any number of participants and define threshold values without limitation. This adaptability enables us to adjust the system to different specifications, such as determining whether signing is limited to high-level nodes or includes lower-level participants as well.

4.2.1 Building Blocks

Cryptographic primitives are crucial in guaranteeing the integrity, validity, and secrecy of data in the rapidly evolving landscape of secure digital communication and decentralized networks. Digital signatures, a fundamental cryptographic mechanism,

allow for the authentication of the origin and integrity of digital messages, thus building confidence in electronic transactions and communications.

Threshold signature schemes have become prominent due to their ability to enhance the security and flexibility of digital signatures. These schemes enable a group of participants to collectively generate signatures while ensuring that a predetermined threshold of participants is required for the signatures to be considered valid. Conventional threshold signatures cannot handle hierarchical patterns commonly found in real-world applications, as they treat all participants equally. One example is bank systems. Consider the management of high-value transactions by either two directors or a director and a president, but not by two presidents. Since the threshold signature scheme is not suitable for this particular scenario, hierarchical threshold signature schemes (HTSS) expand upon the concept of basic threshold signatures, providing a solution that is compatible with hierarchical organizational systems. This section introduces the core concept of access structure and the hierarchical systems that have been proposed in the literature.

4.2.1.1 Access Structure

Access structure is a group of authorized people allowed to reconstruct the secret in secret sharing schemes. Hierarchical access structure consists of two classes: disjunctive and conjunctive access structures. In a hierarchical structure, participants are divided into separate levels based on their significance. These levels maintain a strict hierarchy, with parties in higher levels holding greater importance compared to those in lower levels. In a typical scenario involving a bank's workforce, the higher level might be composed of the board of directors. Simmons [128] introduced the initial hierarchical secret sharing scheme known as disjunctive multilevel secret sharing. Later, Tassa [136] modified this scheme into the conjunctive multilevel secret sharing approach. In both of these schemes, a secret is distributed among participants occupying different authority tiers. The terms "disjunctive" and "conjunctive" were jointly introduced in [14]. To understand the distinction between these two types of access structures, we provide these definitions.

Definition 20 (Access structure [14])**.** *Let U be a set of users. An access structure*

$\Gamma \subseteq P(U)$ *must satisfy these two conditions:*

- *monotonicity: if* $A \in \Gamma$ *and* $A \subseteq B$ *then* $B \in \Gamma$

- *non-triviality: if* $A \in \Gamma$ *then* $|A| > 0$.

If every set A is in Γ, A is authorized and if every set B is not in Γ, B is unauthorized.

Definition 21 (Threshold access structure [14])**.** *We say that* Γ *is a threshold access structure corresponding to threshold* t *if* $\Gamma = \{A \subseteq U : |A| \geq t\}$.

Each level L has a threshold t_L such that $t_0 < t_1 < \cdots < t_n$. For the conjunctive multi-level access structure, if there exists a minimum of t_L users at levels $0, 1, \ldots, L$ *for every level* L, the secret can only be recovered. For a disjunctive multi-level access structure, if *for some level* L, there are at least t_L users at level $0, \ldots, L$, the secret can be recovered.

In a hierarchical secret sharing scheme, a secret α is shared among the players with monotonically increasing thresholds $t_1 < t_2 < \cdots < t_n$. Let P be a set of n players and assume P is composed of l disjoint levels:

$$P = \bigcup_{i=1}^{l} P_i$$

where $P_i \cap P_j = \emptyset$ for all $1 \leq i < j \leq l$ and $|P_i| \geq t_i$ for all i.

Definition 22 (Disjunctive hierarchical access structure [97])**.** *secret* α *can be recovered by a set of players* A*, i.e., an authorized subset, only if*

$$|A \cap (\bigcup_{i=1}^{j} P_i)| \geq t_j \quad \textit{for } \textbf{\textit{at least one}} \; j \textit{ where } 1 \leq j \leq l,$$

i.e., at least one threshold must be satisfied at level 1 *to* j.

Definition 23 (Conjunctive hierarchical access structure [97])**.** *Secret* α *can be re-*

covered by a set of players only if

$$|A \cap (\bigcup_{i=1}^{j} P_i)| \geq t_j \quad \textit{for } \textbf{\textit{all}} \; j \textit{ where } 1 \leq j \leq l,$$

4.2.1.2 Hierarchical Schemes

Starting with the concept of hierarchical secret sharing scheme, several constructions of these schemes for different authorized subsets of participants (access structure) have been proposed in the literature [124, 128, 55, 136, 137, 14, 68, 138, 97, 47, 154]. This section analyze the proposed works in the literature.

In 1979, Shamir [124] pointed out a weighted threshold system as a version of a hierarchical threshold scheme in which participants take the number of shares, which are points on a polynomial, proportional to their level. In this scheme, the ratio of the size of the participant's share to the size of the secret equals the participant's assigned weight, which may be exponential in number of participants [13]. Thus, it can be seen that this construction is not ideal [138].

Simmons [128] introduced the idea of a disjunctive multilevel threshold scheme and compartmented threshold scheme in 1988, both of which were based on a geometric construction Blakley [24] had presented. As in the Blakley threshold scheme, the concept of intersecting hyperplanes is used. To keep the unqualified set from finding the secret, the proposed scheme is not good at reconstructing the secret because it requires the dealer to check the nonsingularity of an exponentially large number of matrices. Further, the scheme is not ideal [68].

In 1989, Brickell [27] introduced an ideal multi-threshold secret-sharing scheme. Nonetheless, this scheme is inefficient, as it necessitates the dealer to compute an exponential number of matrices to guarantee the non-singularity of matrices. Afterward, in 1998, Ghodosi [55] presented a scheme designed for compartmented access structures by applying the Shamir secret sharing scheme. Each level in this scheme has its own polynomial; however, the degrees of these polynomials are recursively defined. Since new participants cannot be added to any level except the last without resharing the secret, the scheme is not dynamic [153]. Also, the proposed scheme

only works for a small number of shareholders [14].

In 2007, Tassa [136] proposed a conjunctive hierarchical scheme based on Birkhoff interpolation. In Tassa's scheme, polynomial derivatives are used to generate shares for participants of lower levels in the hierarchy. To generate the additional shares, the scheme uses the vector of coefficients of the polynomial, which is denoted by $a = (a_0, a_1, \ldots, a_{t-1})$. The dealer then takes a derivative of it to generate a new vector of coefficients for the distribution according to participant indexes, which is denoted by $a' = (a_1, 2 \cdot a_2, \ldots, (t-1) \cdot a_{t-2})$. The derivative operator can be applied $t-1$ times to generate different hierarchy levels. The distributed key generation, which is based on Tassa's protocol and does not require a dealer, is also available [100]. Later, in 2009, Tassa and Dyn [137] proposed an ideal hierarchical secret-sharing scheme based on a bivariate interpolation technique. However, the proposed schemes ([136, 137]) necessitate a large finite field with some limitations in the assigned identities of the users [14]. In the rest of the paper, Tassa's scheme will refer to the first construction [136]. It is important to note that Tassa's technique, whose matrix needs to satisfy necessary Polya's condition, does not necessarily exist in all scenarios and requires exponential complexity due to the matrices' nonsingularity check [154]. After selecting the polynomial at the first level, it is not possible to increase the number of levels in sub-hierarchies, thus this technique might not be appropriate for all kinds of secrets or access hierarchies.

In 2008, Belenkiy [14] proposed disjunctive multi-level secret sharing in which the users learn a point on a polynomial or in its derivative as in Tassa. But instead of classically choosing a constant number as the secret, the secret is chosen as a_{t-1} when the polynomial coefficients are $a = (a_0, a_1, \ldots, a_{t-1})$. She showed that this technique can be used directly to construct disjunctive secret sharing, without the need for an intermediary conjunctive scheme. However, the scheme is not suitable for all scenarios and is not very efficient.

In 2009, Käsper et al. [68] proposed strongly multiplicative hierarchical threshold secret sharing in which players who have a share of the secret take the multiplication of these secret shares without knowing the original secrets, even if the active adversary is present. They gave an efficient linear secret sharing construction for a hierarchi-

cal scheme contrary to the exponential construction in general linear secret sharing scheme. However, their scheme requires stronger conditions on the access structure [141].

In 2013, Tentu et al. [138] introduced a computationally perfect, conjunctive hierarchical scheme based on the maximum distance separable (MDS) codes. In the scheme, the dealer selects MDS codes, its codewords, and a distinct one-way function, then each participant takes exactly one share. The codewords are chosen in a way that they contain shared secrets in its first component, next contain images of shares of relative level participants under distinct one-way functions, and the rest are chosen arbitrarily. This scheme doesn't require the ground field to be extremely large, or any restrictions on the assigned identities of the users. While they pointed out that the idea is also applicable to the disjunctive scheme, they just present a conjunctive scheme in their work.

In 2015, Nojoumian and Stinson [97] introduced sequential secret sharing in which a group of players with varying levels share different yet related secrets with increasing thresholds. Multiple secrets are derived in this protocol by a linear combination of previous secrets. During the reconstruction phase, each subgroup of players can only recover secrets at their designated level. Consequently, the master secret can only be revealed if all the secrets in the higher levels are sequentially recovered. Considering FROST, since the secret needs to be disclosed for signature verification, it cannot be used for threshold signatures.

In 2016, Ersoy et al.[47] presented a new disjunctive and conjunctive multilevel secret-sharing scheme (SSS) built around the anchor sequence, a unique primary sequence. The literature's CRT-based multilevel threshold SSS of Harn-Fuyou [59] is not applicable to all threshold configurations. Ersoy et al. presented their work and revealed that these new schemes can be integrated into function-sharing schemes. However, the scheme employs sequences of primes that could be widely spaced. Producing sequences of prime numbers that satisfy these conditions is resource-intensive. The scheme relies on hash functions that must be treated as random oracles for security purposes, and the information rate is notably high.

In 2022, Yuan et al. [154] presented a new hierarchical scheme based on linear ho-

mogeneous recurrence (LHR) relations. In this scheme, linearly independent homogeneous recurrence relations are chosen by the distributor, and then pseudo shares of the participants from related levels are used to construct the related LHR relation. In the reconstruction, a qualified subset of participants solve relations to get the required values, and then obtain shared secrets. The complexity of hierarchical schemes is reduced from exponential to polynomial in this scheme. However, the participants are assumed to be semi-honest, and there is no verification check in their scheme.

In 2023, Ağırtaş and Yayla [5] proposed compartment-based and hierarchically delegated signing authority within the verifiable accountable subgroup multi-signature (vASM) framework. In this framework, the authorized user delegates the signing power of his membership key in vASM to a certain number of unauthorized users with different constructions in recursive vASM, Shamir's secret sharing (direct and nested way), and Käsper's hierarchical threshold secret sharing. Then the message is signed by the sum of the unauthorized user's secret key and the authorized user's compartment-based membership key. These constructions give a general delegation that doesn't have a warrant to provide more comprehensive details about the delegation.

4.2.2 The Proposed Flexible Hierarchical Threshold Signature Scheme: FlexHi

This section presents a novel hierarchical threshold signature scheme called a flexible hierarchical threshold signature (FlexHi) scheme. FlexHi utilizes independent polynomials at each level, offering the flexibility to allow any number of participants and set threshold values without limitations as opposed to the monotone threshold requirement in conjunctive or disjunctive hierarchical-based schemes. This flexibility ensures that the scheme can adapt to various hierarchical structures and security requirements, making it a versatile and robust solution for a hierarchical threshold signature.

Definition 24 (Flexible hierarchical secret sharing)**.** *Let $\mathcal{L}$ denote a hierarchical structure comprising multiple levels such that*

$$\mathcal{L} = \bigcup_{j=1}^{n} \mathcal{L}_j$$

where $\mathcal{L}_j \cap \mathcal{L}_k = \mathcal{L}_j$ *for all* $1 \leq j < k \leq n$. *These levels are listed from the highest* ($\mathcal{L}_1$) *to the lowest* ($\mathcal{L}_n$) *hierarchy level where each lower level includes all the nodes from the upper levels. Each level* $\mathcal{L}_j(t_j, m_j)$ *consists of* $\bigcup_{s=1}^{j} \mathcal{N}_{s,i}$ *nodes where* $1 \leq i \leq m_j$ *and* t_j *is the threshold value.*

In FlexHi scheme, a private key share sk_j can be constructed when at least t_j nodes $\mathcal{N}_{j,i} \in \mathcal{L}_j(t_j, m_j)$ are engaged in a key construction algorithm (see Figure 4.10). Then public key shares are generated from the corresponding private key shares depending on the underlying signature scheme. These independent level-based public keys are aggregated to generate the main public key. The hierarchical node has the advantage of extra secrets over lower-level nodes, and the scheme requires the use of this secret.

Example 1. Suppose the aim is to create a three-level flex hierarchical threshold signature scheme with thresholds[2] $t_1 = 2$, $t_2 = 2$, and $t_3 = 6$ where $\mathcal{L}_1(2, 3)$, $\mathcal{L}_2(2, 6)$, and $\mathcal{L}_3(6, 9)$. Let the set of nodes be:

$$\mathcal{L}_1 = \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}\}$$
$$\mathcal{L}_2 = \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}, \mathcal{N}_{2,1}, \mathcal{N}_{2,2}, \mathcal{N}_{2,3}\}$$
$$\mathcal{L}_3 = \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}, \mathcal{N}_{2,1}, \mathcal{N}_{2,2}, \mathcal{N}_{2,3}, \mathcal{N}_{3,1}, \mathcal{N}_{3,2}, \mathcal{N}_{3,3}\}$$

KeyGen:

1. **($\mathcal{L}_1(2, 3)$).** Each node $\mathcal{N}_{1,i} \in L_1$, where $1 \leq i \leq 3$:

   (a) Uses secret sampling in Figure 4.10 to create his secret $s_{11,i}$ for $\mathcal{L}_1$.

   (b) Adds the related subshares obtained in $\mathcal{L}_1$ to construct his private key share $sk_{11,i}$ using key construction in Figure 4.10. In this construction,

[2] In our scheme, there is no requirement for threshold values to be in a specific order, whether monotonous increasing or decreasing. Without loss of generality, these values can be chosen arbitrarily depending on the application e.g. $t_1 = 2$, $t_2 = 4$, and $t_3 = 3$.

public key share $pk_{11,i}$ is also created from the underlying signature scheme, and it is made public.

(c) Generates the first level public key pk_1 from all published public key shares.

2. **($\mathcal{L}_2(2,6)$).** Each node $\mathcal{N}_{1,i}, \mathcal{N}_{2,i} \in \mathcal{L}_2$, where $1 \leq i \leq 3$:

(a) Uses secret sampling in Figure 4.10 to create his secret $s_{21,i}$,$s_{22,i}$, respectively, for $\mathcal{L}_2$.

(b) Adds the related subshares received in $\mathcal{L}_2$ to construct his private key share $sk_{21,i}, sk_{22,i}$, respectively, using key construction in Figure 4.10. In this construction, public key share $pk_{21,i}, pk_{22,i}$, respectively, is also created from the underlying signature scheme, and they are made public.

(c) Generates the second level public key pk_2 from all published public signing key shares

3. **($\mathcal{L}_3(6,9)$).** Each participant $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}, \mathcal{N}_{3,i} \in \mathcal{L}_3$, where $1 \leq i \leq 3$:

(a) Uses secret sampling in Figure 4.10 to create his secret $s_{31,i}, s_{32,i}, s_{33,i}$, respectively, for $\mathcal{L}_3$.

(b) Adds the related subshares received in $\mathcal{L}_3$ to construct his private key share $sk_{31,i}, sk_{32,i}, sk_{33,i}$, respectively, using key construction in Figure 4.10. In this construction, public key share $pk_{31,i}, pk_{32,i}, pk_{33,i}$ respectively, is also created from the underlying signature scheme, and they are made public.

(c) Generates the third level public key pk_3 from all published public signing key shares

KeyAgg All level-based public keys pk_1, pk_2, pk_3 are combined to generate the main public key pk

SignGen

1. **($\mathcal{L}_1(2,3)$).** Any two of $\mathcal{N}_{1,i}$ use his private key share $sk_{11,i}$ to sign the message partially as $\sigma_{11,i}$, where $1 \leq i \leq 3$

2. **($\mathcal{L}_2(2,6)$).** Any two of $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}$ use his private key share $sk_{21,i}, sk_{22,i}$, respectively, to sign the message partially as $\sigma_{21,i}, \sigma_{22,i}$, where $1 \leq i \leq 3$.

3. **($\mathcal{L}_3(6,9)$).** Any six of $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}, \mathcal{N}_{3,i}$ use his private key share $sk_{31,i}, sk_{32,i}$, $sk_{33,i}$, respectively, to sign the message partially $\sigma_{31,i}, \sigma_{32,i}, \sigma_{33,i}$, where $1 \leq i \leq 3$.

SignAgg The quorum partial signatures generate the main signature σ :

$$\{\sigma_{11,i} \in \mathcal{L}_1\}^{t_1=2} \wedge \{\sigma_{21,i} \in \mathcal{L}_2 \vee \sigma_{22,i} \in \mathcal{L}_2\}^{t_2=2}$$
$$\wedge\{\sigma_{31,i} \in \mathcal{L}_3 \vee \sigma_{32,i} \in \mathcal{L}_3 \vee \sigma_{33,i} \in \mathcal{L}_3\}^{t_3=6}$$

where $1 \leq i \leq 3$, $\wedge$ represents “and", $\vee$ represents “or" notation

The formalized notion of FlexHi scheme uses the $\mathcal{F}_{FlexHi}$ function. FlexHi scheme consists of a tuple of six polynomial time algorithms, $\mathcal{F}_{FlexHi} = (\texttt{Setup, KeyGen, KeyAgg, SignGen, SignAgg,Verif})$ as introduced in Figure 4.9.

Flex Hierarchical Threshold Signature Scheme Functionality $\mathcal{F}_{FlexHi}$

Setup: On the input of the security parameter 1^λ, public parameters pp are generated according to the underlying signature scheme.

KeyGen: Taking public parameters pp, each node $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$, where $1 \leq s \leq n$:

- Runs the KeyGen as defined in Figure 4.10 to generate a public/private key shares $(pk_{s,i}, sk_{s,i})$.
- Stores $(pk_{s,i}, sk_{s,i})$ and send $pk_{s,i}$ to the other nodes in that level.
- Computes level's public key pk_s by combining all received public key shares.
- Sets an internal flag ready to 1 and ignore further calls.

KeyAgg: Taking different level's public keys pk_s from each level, do:

- Run the key aggregation algorithm in Figure 4.13, and construct the main public key pk.

SignGen: Taking (sid, m) from each node $\mathcal{N}_{s,l}$ in the same level $\mathcal{L}_s$, $sid = (\mathcal{N}_{s,l}, sid')$ for some sid', if ready $= 1$ and the session identifier sid has not been used previously, then each node $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$:

- Generates partial signature $\sigma_{s,i}$ on message m using the corresponding private key share $sk_{s,i}$, then sends it to each node $\mathcal{N}_{s,l}$ in the same level $\mathcal{L}_s$.
- Stores internally $(sid, \mathsf{delivered})$
- Computes level's signature σ_s after receiving all partial signatures in that level $\mathcal{L}_s$.

SignAgg: Taking different level's partial signatures σ_s from each level, construct the main signature σ.

Verif: If $\mathsf{Verify}(\sigma, \mathsf{m}) = 1$, then the verification is done.

Figure 4.9: Flex Hierarchical Threshold Signature Scheme Functionality $\mathcal{F}_{FlexHi}$

The KeyGen procedure in the Figure 4.10 algorithm consists of two rounds. In Round 1, nodes select secrets, create secret polynomials, generate knowledge proofs,

and verify these proofs to ensure the validity of the secrets. After secret polynomials are constructed, in Round 2, nodes compute subshares and use them to calculate private key shares. They also generate public key shares using the underlying signature scheme. This process establishes the keys for the algorithm.

`KeyGen` of $\mathcal{F}_{FlexHi}$

Round 1. (**Secret Sampling**) On input security parameter 1^λ, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ does the followings ($1 \leq i \leq m_s$):

- Takes a secret $s_{s,i} \in F_q$
- Creates a polynomial $f_{s,i}(x) = s_{s,i} + \sum_{i=1}^{t-1} a_i.x_i$ where $a_i \in F_q$ is the randomly chosen coefficients of the polynomial
- Runs `ProofGen`$(1^\lambda, id_{s,i}, g^{s_{s,i}})$ algorithm in Figure 4.11 to prove knowledge of secret by outputting $PoK(s_{s,i})$, then broadcast it to all $\mathcal{N}_{s,l}$ node in the same level $\mathcal{L}_s$ ($1 \leq l \leq m_s, l \neq i$)
- Runs `ProofVerify`$(1^\lambda, id_{s,i}, PoK(s_{s,i}))$ algorithm in Figure 4.12 to verify the knowledge of secret $s_{s,i}$

Round 2. (**Key Construction**)

After constructing secret polynomial $f_{s,i}(x)$ from Round 1, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ does the followings ($1 \leq i \leq m_s$):

- Computes subshares $ss_{i,l}$ for $\mathcal{N}_{s,l}$ by evaluating a point on polynomial $f_{s,i}(l)$, then sends these computed values to $\mathcal{N}_{s,l}$
- When receiving $(m_s - 1)$ subshares, computes its private key share $sk_{s,i}$ by adding related subshares $ss_{l,i}$ where $1 \leq l \leq m_s$. Public key share $pk_{s,i}$ is also generated from $sk_{s,i}$ using the underlying signature scheme.

Figure 4.10: `KeyGen` of $\mathcal{F}_{FlexHi}$

The `ProofGen` procedure in Figure 4.11 algorithm is responsible for generating a proof of knowledge $PoK(s_{s,i})$ for a specific secret $s_{s,i} \in F_q$. This process is carried out by each node $\mathcal{N}_{s,i}$ within the same level $\mathcal{L}_s$. The output ensures that the node possesses the knowledge of the secret $s_{s,i}$ without revealing it. This proof is verified by the `ProofVerify` procedure in Figure 4.12. The `KeyAgg` procedure in Figure 4.13 combines level-based public keys to generate the main public key.

`ProofGen` of $\mathcal{F}_{FlexHi}$

On input security parameter 1^λ, secret in commitment $g^{s_{s,i}}$, and id $id_{s,i}$, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ ($1 \leq i \leq m_s$) calculates a proof of knowledge $PoK(s_{s,i})$ to corresponding secret $s_{s,i} \in F_q$

Figure 4.11: `ProofGen` of $\mathcal{F}_{FlexHi}$

`ProofVerify` of $\mathcal{F}_{FlexHi}$

On input security parameter 1^λ, a proof of knowledge $PoK(s_{s,i})$, and id $id_{s,i}$, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ ($1 \leq i \leq m_s$) verifies a proof of knowledge of secret

Figure 4.12: `ProofVerify` of $\mathcal{F}_{FlexHi}$

`KeyAgg` of $\mathcal{F}_{FlexHi}$

The level-based public keys pk_s $(1 \leq s \leq n)$ are combined to generate the main public key $pk = pk_1 * pk_2 * \cdots * pk_n$ where $*$ denotes the combination operator.

Figure 4.13: `KeyAgg` of $\mathcal{F}_{FlexHi}$

4.2.3 FlexHi FROST Scheme Application

This section presents the proposed FlexHi scheme application to the FROST (Flexible Round-Optimized Schnorr Threshold) signature scheme. FROST [72] is a round-optimized threshold signature scheme based on the Schnorr signature [122], which enhances robustness by allowing a quorum of honest participants to identify instances of misbehavior. FROST includes a semi-trusted role in the final stage of the signature, which is referred to as the signature aggregator $\mathcal{SA}$. The purpose of $\mathcal{SA}$ is to minimize communication between participants, thus it can be also implemented without the need for a $\mathcal{SA}$. Adhering to the algorithm in FROST and the proposed hierarchical model is applied to FROST. The applied scenario consists of two levels $\mathcal{L}_1(1,1)$ and $\mathcal{L}_2(t,m)$. The first level is hierarchically strong and has one hierarchical node, the other is the level with m nodes and t threshold value.

Key Generation Stage. The key generation phase of FlexHi FROST application is given in Figure 4.14. The key generation of FROST is based on Pedersen's Distributed Key Generation [102]. In addition to Pedersen key generation, participants must prove the constant term of the polynomials (secret) they produce with zero-knowledge proofs to prevent rogue key attacks [15] in which attackers are allowed to arbitrarily choose their public keys. Since $\mathcal{L}_1(1,1)$ consists of a single node $\mathcal{N}_{1,1}$ in our scenario, it does not need key generation rounds at his level. It will be enough for him to generate the private key himself and publish its public key. However, as stated in the previous section, $\mathcal{N}_{1,1}$ needs to join the key generation of $\mathcal{L}_2(t,m)$. In this case, $\mathcal{N}_{1,1}$ has an extra secret. On the other hand, the nodes in $\mathcal{L}_2(t,m)$ should run the protocol as is, adhering to the `KeyGen` phase in FROST. Each node $\mathcal{N}_{2,i}$ randomly selects a polynomial of degree t and creates a Schnorr Proof for its constant term by running `ProofGen`. They then broadcast the proof values and the commitment of the coefficients of the polynomial. Nodes verify proof values from other

FlexHi `KeyGen` Procedure for FROST Scheme

For $\mathcal{L}_1(1,1)$:

1. $\mathcal{N}_{1,1}$ samples a random private key $sk_1 \xleftarrow{\$} \mathbb{Z}_q$
2. $\mathcal{N}_{1,1}$ computes level public key $pk_1 = g^{sk_1}$

For $\mathcal{L}_2(t,m)$:

Round 1 For $1 \leq i \leq m$,

1. Every node $\mathcal{N}_{2,i}$ chooses t random values $\left(a_{2i,0}, \ldots, a_{2i,(t-1))}\right) \xleftarrow{\$} \mathbb{Z}_q$, and uses them as coefficients to define a degree $t-1$ polynomial $f_{2i}(x) = a_{2i,0} + \sum_{j=1}^{t-1} a_{2i,j}x^j$ where $a_{2i,0}$ is chosen secret.
2. Node $\mathcal{N}_{2,i}$ runs $\texttt{ProofGen}(1^\lambda, id_{2i}, \Phi, g^{a_{2i,0}})$ in Figure 4.15 for $\mathcal{L}_2(t,m)$ to demonstrate $PoK(a_{2i,0})$ by outputting κ_{2i} and commitment $\vec{C}_{2i}$.
3. Upon receiving $\vec{C}_{2\ell}, \kappa_{2\ell}$ from other participants $1 \leq \ell \leq n$, $\ell \neq i$, Node $\mathcal{N}_{2,i}$ runs $\texttt{ProofVerify}(1^\lambda, id_{2\ell}, \Phi, \kappa_{2\ell}, \vec{C}_{2\ell})$ in Figure 4.16 for $L_2(t,m)$ to verify proof of their corresponding secret.

Round 2 For $1 \leq i \leq m$,

1. Each node $\mathcal{N}_{2,i}$ securely sends to each other level 2 node $\mathcal{N}_{2\ell}$ a secret share (subshare) $(2\ell, f_{2i}(id_{2\ell}))$, deleting f_{2i} and each share afterward except for $(id_{2i}, f_{2i}(id_{2i}))$ which they keep for themselves.
2. Each node $\mathcal{N}_{2,i}$ verifies their shares by calculating: $g^{f_{2\ell}(id_{2i})} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{2\ell,k}^{{id_{2i}}^k} \bmod q$, aborting if the check fails.
3. Each node $\mathcal{N}_{2,i}$ calculates their long-lived private key share by computing $sk_{2i} = \sum_{\ell=1}^{n} f_{2\ell}(id_{2i})$, stores sk_{2i} securely, and deletes each $f_{2\ell}(i)$.
4. Each node $\mathcal{N}_{2,i}$ calculates their public verification share $pk_{2i} = g^{sk_{2i}}$, and the level's public key $pk_2 = \prod_{j=1}^{m} \phi_{2,j,0}$. Any participant can compute the public verification share of any other participant by calculating

$$pk_{2i} = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \phi_{2,j,k}^{id_{2i}^k \bmod q}.$$

Figure 4.14: FlexHi `KeyGen` Procedure for FROST Scheme

FlexHi `ProofGen` Procedure for FROST Scheme

For $\mathcal{L}_1(1,1)$:

1. $\mathcal{N}_{1,1}$ calculates a proof of knowledge to the corresponding private key sk_1 by calculating $\kappa_1 = (R_1, \mu_1)$, such that $r_1 \xleftarrow{\$} \mathbb{Z}_q$, $R_1 = g^{r_1}$, $c_1 = H\left(id_{1,1}, \Phi, g^{sk_1}, R_1\right)$, $\mu_1 = r_1 + sk_1 \cdot c_1$ with Φ being a context string to prevent replay attacks. Note that $\mathcal{N}_{1,1}$ has already computed commitment g^{sk_1} as Pk_1 value.

For $\mathcal{L}_2(t,m)$:

1. Every node $\mathcal{N}_{2,i}$ calculates a proof of knowledge to the corresponding secret $a_{2i,0}$ by calculating $\kappa_{2i} = (R_{2i}, \mu_{2i})$, such that $r \xleftarrow{\$} \mathbb{Z}_q$, $R_{2i} = g^r$, $c_{2i} = H\left(id_{2i}, \Phi, g^{a_{2i,0}}, R_{2i}\right)$, $\mu_{2i} = r + a_{2i,0} \cdot c_{2i}$ with Φ being a context string to prevent replay attacks.
2. Also, every node $\mathcal{N}_{2,i}$ calculates a commitment $\vec{C}_{2i} = \left\langle \phi_{2i,0}, \ldots, \phi_{2i,(t-1)} \right\rangle$ such that $\phi_{2i,j} = g^{a_{2i,j}}$, where $0 \leq j \leq t-1$.

Figure 4.15: FlexHi `ProofGen` Procedure for FROST Scheme

FlexHi `ProofVerify` Procedure for FROST Scheme

For $\mathcal{L}_1(1,1)$:

1. Upon receiving pk_1, κ_1 from $\mathcal{N}_{1,1}$, each level 2 nodes $1 \leq i \leq m$, $\mathcal{N}_{2,i}$ verifies $\kappa_1 = (R_1, \mu_1)$, aborting on failure, by checking $R_1 \stackrel{?}{=} g^{\mu_1} \cdot pk_1^{-c_1}$, where $c_1 = H\left(id_{1,1}, \Phi, g^{sk_1}, R_1\right)$.

For $\mathcal{L}_2(t,m)$:

1. Upon receiving $\vec{C}_{2\ell}, \kappa_{2\ell}$ from level 2 nodes $1 \leq \ell \leq m, \ell \neq i$, node $\mathcal{N}_{2,i}$ verifies $\kappa_{2\ell} = (R_{2\ell}, \mu_{2\ell})$, aborting on failure, by checking $R_{2\ell} \stackrel{?}{=} g^{\mu_{2\ell}} \cdot \phi_{2\ell,0}^{-c_{2\ell}}$, where $c_{2\ell} = H\left(id_{2\ell}, \Phi, \phi_{2\ell,0}, R_{2\ell}\right)$.

Figure 4.16: FlexHi `ProofVerify` Procedure for FROST Scheme

nodes by running `ProofVerify`. If there is no error during verification, each node sends a subshare of secret values to other nodes. After that, each node $\mathcal{N}_{2,i}$ creates its private key share using incoming subshares. After these steps, public key share, and level-based public key is calculated.

FlexHi `KeyAgg` Procedure for FROST Scheme

1. Node $\mathcal{N}_{1,1}$ runs `ProofGen`$(1^\lambda, id_{1,1}, \Phi, g^{sk_1})$ for $\mathcal{L}_1(1,1)$ that outputs κ_1 to demonstrate zero knowledge proof of corresponding private key sk_1, then broadcasts κ_1 and pk_1 to all other level 2 nodes.
2. Node $\mathcal{N}_{2,i}$ runs `ProofVerify`$(1^\lambda, id_{1,1}, \Phi, \kappa_1, pk_1)$ for $L_1(1,1)$ to verify zero knowledge proof of corresponding private key sk_1.
3. After the proof verification, each level 2 nodes $1 \leq i \leq m$, $\mathcal{N}_{2,i}$ sends pk_{2i} to $\mathcal{N}_{1,1}$.
4. For $1 \leq i \leq m$, each level 2 nodes $\mathcal{N}_{2,i}$ and $\mathcal{N}_{1,1}$ compute main public key

$$pk = pk_1 * pk_2$$

Figure 4.17: FlexHi `KeyAgg` Procedure for FROST Scheme

Key Aggregation Stage. In the key aggregation algorithm of FlexHi FROST application that is given in Figure 4.17, the main public key pk is generated with a multiplication operator taken as a combination operator.

Preprocessing Stage. The preprocessing stage of FlexHi FROST application in Figure 4.18 is performed before the signing operation. In this stage, every node at every level generates a set of π random number pairs, which are one-time use private nonces and their corresponding commitment shares. Each node calculates its own commitment shares using these number pairs and shares them with the other participants. The value of π determines how many signing operations can be performed before the next preprocess stage.

Signing Stage. In the signing phase of our FROST application, $\mathcal{N}_{1,1}$ and each level

FlexHi `Preprocessing` Procedure for FROST Scheme

For $\mathcal{L}_1(1,1)$**:**

1. $\mathcal{N}_{1,1}$ creates an empty list $\mathcal{L}_{11}$. Then, for $1 \leq j \leq \pi$ where j be a counter for a specific commitment share pair, and π be the number of pairs generated at a time perform the following:
   (a) Sample single-use nonces $(d_{11,j}, e_{11,j}) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
   (b) Derive commitment shares $(D_{11,j}, E_{11,j}) = \left(g^{d_{11,j}}, g^{e_{11,j}}\right)$.
   (c) Append $(D_{11,j}, E_{11,j})$ to L_{11}.
   (d) Store $\left((d_{11,j}, D_{11,j}), (e_{11,j}, E_{11,j})\right)$ for later use in signing operations.
2. Publish $(id_{1,1}, L_{11})$ to a predetermined location.

For $\mathcal{L}_2(t,m)$**:**

1. For $1 \leq i \leq m$, every node $\mathcal{N}_{2,i}$ creates an empty list $\mathcal{L}_{2i}$. Then, for $1 \leq j \leq \pi$ where j be a counter for a specific commitment share pair, and π be the number of pairs generated at a time perform the following:
   (a) Sample single-use nonces $(d_{2i,j}, e_{2i,j}) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
   (b) Derive commitment shares $(D_{2i,j}, E_{2i,j}) = \left(g^{d_{2i,j}}, g^{e_{2i,j}}\right)$.
   (c) Append $(D_{2i,j}, E_{2i,j})$ to L_{2i}.
   (d) Store $\left((d_{2i,j}, D_{2i,j}), (e_{2i,j}, E_{2i,j})\right)$ for later use in signing operations.
2. Publish $(id_{2i}, \mathcal{L}_{2i})$ to a predetermined location.

Figure 4.18: FlexHi `Preprocessing` Procedure for FROST Scheme

2 node $\mathcal{N}_{2,i}$ perform similar operations as defined in Figure 4.19. Each node creates same group commitment by using the nonces they choose during the preprocessing phase and creates a partial signature by processing them with their private keys. Finally, they send these signatures to $\mathcal{SA}$.

Signature Aggregation Stage. In the signature aggregation phase defined in Figure 4.20, $\mathcal{SA}$ verifies each signature and creates the signature of the level. Finally,$\mathcal{SA}$ creates the signature by combining these two signatures.

Signature Verification Stage. As in the standard Schnorr's verification [122] operation, the main signature $\sigma = (R, z)$ on message m is verifiable with main public key pk.

4.2.4 System Analysis

System analysis comprises two essential components: security and performance analysis of FlexHi scheme. The security analysis focuses on evaluating the indistinguisha-

FlexHi `SignGen` Procedure for FROST Scheme

Let S_2 be set of $\alpha : t \leq \alpha \leq n$ participants that are selected for signing in second level threshold scheme. Let $B = \langle (id_{2i}, id_{1,1}, D_{2i}, E_{2i}, D_{11}, E_{11}) \rangle_{i \in S_2}$ denote the ordered list of node indices corresponding to each node $\mathcal{N}_{s,i}$. Let H_1, H_2 be hash functions whose outputs are in $\mathbb{Z}_q^*$

- For each $i \in S$, $\mathcal{SA}$ sends $\mathcal{N}_{2,i}$ and $\mathcal{N}_{1,1}$ the tuple (m, B)

For $\mathcal{L}_1(1,1)$:

1. After receiving (m, B), $\mathcal{N}_{1,1}$ first validates the message m, and then checks $D_{11}, E_{11} \in \mathbb{G}^*$ in B, aborting if either check fails.

2. $\mathcal{N}_{1,1}$ computes the set of binding values $\rho_{2\ell} = H_1(id_{2\ell}, m, B)$, and $\rho_{11} = H_1(id_{1,1}, m, B)$, $\ell \in S_2$. $\mathcal{N}_{1,1}$ then derives the group commitment $R = \prod_{\ell \in S_2} D_{2\ell} \cdot D_{11} \cdot (E_{2\ell})^{\rho_{2\ell}} \cdot (E_{11})^{\rho_{11}}$, and the challenge $c = H_2(R, pk, m)$.

3. Using the first level private key sk_1, $\mathcal{N}_{1,1}$ computes $z_1 = d_{11} + (e_{11} \cdot \rho_{11}) + sk_1 \cdot c$,

4. $\mathcal{N}_{1,1}$ securely deletes $((d_{11}, D_{11}), (e_{11}, E_{11}))$ from the local storage, and returns z_1 to $\mathcal{SA}$.

For $\mathcal{L}_2(t,m)$:

1. After receiving (m, B), each $\mathcal{N}_{2,i}$ first validates the message m, and then checks $D_{2\ell}, E_{2\ell} \in \mathbb{G}^*$ for each commitment in B, aborting if either check fails.

2. Each $\mathcal{N}_{2,i}$ then computes the set of binding values $\rho_{2\ell} = H_1(id_{2\ell}, m, B)$, and $\rho_{11} = H_1(id_{1,1}, m, B)$, $\ell \in S_2$. Each $\mathcal{N}_{2,i}$ then derives the group commitment $R = \prod_{\ell \in S_2} D_{2\ell} \cdot D_{1,1} \cdot (E_{2\ell})^{\rho_{2\ell}} \cdot (E_{11})^{\rho_{11}}$ and the challenge $c = H_2(R, pk, m)$.

3. Each $\mathcal{N}_{2,i}$ computes their response using their long-lived private key share sk_{2i} by computing $z_{2i} = d_{2i} + (e_{2i} \cdot \rho_{2i}) + \lambda_{2i} \cdot sk_{2i} \cdot c$, using S_2 to determine the i^{th} Lagrange coefficient λ_{2i}.

4. Each $\mathcal{N}_{2,i}$ securely deletes $((d_{2i}, D_{2i}), (e_{2i}, E_{2i}))$ from their local storage, and then returns z_{2i} to $\mathcal{SA}$.

Figure 4.19: FlexHi `SignGen` Procedure for FROST Scheme

FlexHi `SignAgg` Procedure for FROST Scheme

1. $\mathcal{SA}$ derives $\rho_{2i} = H_1(id_{2i}, m, B)$, $\rho_{11} = H_1(id_{1,1}, m, B)$, $R_{2i} = D_{2i,j} \cdot (E_{2i,j})^{\rho_{2i}}$, and $R_{11} = D_{11,j} \cdot (E_{11,j})^{\rho_{11}}$ for $i \in S$. Subsequently, $\mathcal{SA}$ derives $R = \prod_{i \in S} R_{2i} \cdot R_{11}$ and $c = H_2(R, pk, m)$.

2. $\mathcal{SA}$ verifies the validity of each response by checking $g^{z_{2i}} \stackrel{?}{=} R_{2i} \cdot pk_{2i}^{c \cdot \lambda_{2i}}$ for each signing share $z_{2i}, i \in S$. Also, $g^{z_1} \stackrel{?}{=} R_{11} \cdot pk_{11}^{c}$ is checked. If the equality does not hold, identify and report the misbehaving node and then abort. Otherwise, continue.

3. $\mathcal{SA}$ computes the level's response $z_2 = \sum z_{2i}$

4. $\mathcal{SA}$ computes $z = z_1 + z_2$, then broadcasts the main signature as $\sigma = (R, z)$ along with m.

Figure 4.20: FlexHi `SignAgg` Procedure for FROST Scheme

bility and unforgeability aspects of the scheme using game-based security. Additionally, the analysis addresses the ideality, perfectness, and resistance to collusion attacks. The performance analysis evaluates the efficiency of the FlexHi scheme in comparison to the basic FROST protocol and Tassa's hierarchical threshold signature in the FROST application.

4.2.4.1 Security Analysis

Definition 25 (Lagrange interpolation [12])**.** *For every field $\mathbb{F}$, and a given t different points (x_i, y_i), $1 \leq i \leq t$, there exits a unique polynomial P of degree at most $t-1$ over $\mathbb{F}$ such that $P(x_i) = y_i$, $1 \leq i \leq t$.*

The Lagrange interpolating polynomial is calculated as follows:

$$P(x) = \sum_{i=1}^{t} P_i(x) \ , \ P_i(x) = y_i. \prod_{j=1, j\neq i}^{t} \frac{x - x_j}{x_i - x_j}.$$

The indistinguishability and unforgeability properties of the FlexHi scheme are demonstrated by employing game-based security, where the adversary is represented as a polynomial time algorithm. Let λ represent the security parameter, and $\mathsf{negl}(\lambda)$ denote the negligible success probability of an adversary, indicating that it is extremely small and effectively zero [26].

Secret Indistinguishability. Based on the game-based security proofs for secret sharing schemes in [150], the secret indistinguishability of the proposed FlexHi scheme is defined by the following experiment $\mathsf{Exp}_{\mathcal{A}}^{SecInd,b}(\lambda)$ between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$:

$\mathsf{Exp}_{\mathcal{A}}^{SecInd,b}(\lambda)$
For a given field $\mathbb{F}$
$(\mathsf{s}_0, \mathsf{s}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{Choose}}(\mathbb{F})$
For a given $\mathsf{b} \xleftarrow{\$} \{0,1\}$ and
$\mathsf{ss}_{\mathsf{i,j}}^{\mathsf{b}} \leftarrow \mathsf{Share}(\mathsf{s}_{\mathsf{b}})$:
$\mathsf{b}' \leftarrow \mathsf{Guess}(\mathsf{ss}_{\mathsf{i,j}}^{\mathsf{b}}) \wedge \mathsf{b} = \mathsf{b}'$

Figure 4.21: Secret Indistinguishability Experiment of FlexHi Scheme

- The challenger $\mathcal{C}$ generates the field $\mathbb{F}$ as public parameter.

- Adversary $\mathcal{A}$ chooses two distinct but the same length secret s_0 and s_1 in that field, then sends them to the challenger $\mathcal{C}$.

- The challenger $\mathcal{C}$ selects uniformly random bit $\mathsf{b} \xleftarrow{\$} \{0,1\}$, then sends challenge subshare $(\mathsf{ss}_{\mathsf{i,j}}^{\mathsf{b}})_{j \in A'} \leftarrow \mathsf{Share}(\mathsf{s}_{\mathsf{b}})$ to $\mathcal{A}$ using oracle (A' represents an unauthorized subset)

- Adversary $\mathcal{A}$ outputs a guess bit $\mathsf{b}' \in \{0,1\}$

The output of the experiment is defined to be 1 if $\mathsf{b}' = \mathsf{b}$.

Theorem 6. *FlexHi scheme satisfies secret indistinguishability property given that*

$$\left| \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{SecInd},b}(\lambda) = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)$$

Proof. FlexHi scheme uses the key generation of Shamir's secret-sharing scheme without a trusted dealer version [102]. Based on this, in our scheme (see Figure 4.10) each node chooses uniformly random secret s_i, then creates their private key share sk_i by taking all related subshares ss_i. Thus, each node contributes one piece of that private key share. Since the secrets are chosen uniformly randomly, the distribution of subshares is indistinguishable from uniform distribution. □

Unforgeability. Unforgeability is another important security property in signature schemes, demonstrating an adversary's inability to create a valid signature for a message that hasn't been previously signed. In the context of threshold unforgeability, the assumption is that the adversary has compromised $t-1$ servers. This notion is captured by the following game $\mathsf{Game}_{\mathcal{A}}^{Unf}(\lambda)$:

$\mathsf{Game}_{\mathcal{A}}^{Unf}(\lambda)$
For a given public key pk
$\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(m)$ for some m
Output a new pair (σ^*, m^*) :
$\mathsf{Verify}(\sigma^*, \mathsf{pk}) = 1, m^* \neq m$

Figure 4.22: Unforgeability Experiment of FlexHi Scheme

- The challenger $\mathcal{C}$ generates the public key of the scheme, and provides access to signing oracle $\mathcal{O}_{\mathsf{Sign}}$.

- When adversary $\mathcal{A}$ requests a signature σ for some messages m, the challenger $\mathcal{C}$ outputs their signatures.

- Eventually $\mathcal{A}$ generates a new signature σ^* for message m^*

The adversary succeeds in the game if the verification of the signature σ^* for message m^* is correct, but m^* is not the same as any of the queried inputs to the oracle.

Theorem 7. *FlexHi scheme satisfies unforgeability property if the adversary's advantage is negligible*

$$\left|\Pr[\mathsf{Game}_{\mathcal{A}}^{\mathsf{Unf}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. In FlexHi scheme, the message is signed partially by the eligible node's private key share. These shares are constructed from the subshares of the chosen secret in Shamir. Thus, the security of FlexHi scheme is based on Lagrange's interpolation as in Definition 25. According to this, the private key share can be generated by only an authorized subset of participants who hold t points (subshare) of the polynomial P. On the other hand, an authorized subset $t-1$ of participants who hold $t-1$ points (subshare) of the polynomial P cannot generate the secret polynomial, and thus cannot forge private key share. □

Ideality and Perfectness. The most important parameter for the secret-sharing system is the perfectness of the system that is required to protect the secret. Perfect schemes are referred to as unconditionally secure schemes [132]. The other desirable property is the ideality of the scheme. However, it should be noted that for every access structure, we cannot find an ideal scheme [17].

Definition 26 (Perfectness [132])**.** *A secret sharing scheme is a perfect realization of access structure* Γ *if*

- *An authorized subset of participants* $\mathsf{A} \in \Gamma$ *can always reconstruct the secret,*

- *An unauthorized subset $t-1$ of participants $\mathsf{A}' \in \Gamma$ cannot obtain any information about the secret.*

Definition 27 (Ideality [132])**.** *A secret sharing scheme is ideal if the length of the share of all participants is less than or equal to the size of the secret.*

Lemma 8. *The proposed scheme is not ideal but it is perfect.*

Proof. In FlexHi scheme, the secret is divided into shares for each level, and the shares in each level generate the polynomial of that level. Each level's polynomial is based on Shamir secret sharing which is perfect since its coefficient matrix is a square Vandermonde matrix and it is always nonsingular [153]. This makes the proposed FlexHi scheme is also perfect. On the other hand, FlexHi scheme uses a general flexible access structure that allows us to choose which subset of participants can reconstruct the secret. However, since each participant takes a certain number of shares in proportion to their levels, the length of the shares at a higher level becomes larger than the secret. For this reason, the general flexible access structure of FlexHi scheme cannot be expected to reach the ideal. □

Collusion attack resistance. In FlexHi scheme, the shares of secret signing keys are maintained by level-based polynomials. However, there is no correlation between each of the level-based polynomials. Considering a collusion attack, the attackers need to get all level's polynomials carrying the secret key shares but this is not practical in a real-world scenario. Thus, FlexHi scheme provides resistance against collusion attacks.

4.2.4.2 Performance Analysis

The efficiency of the FlexHi scheme is compared to the basic threshold FROST scheme [72], and the overhead imposed by the hierarchical structure is analyzed. Furthermore, the FlexHi scheme is compared to Tassa's hierarchical threshold signature scheme [136] when applied to the FROST scheme. Tassa's pioneering work is rooted in the construction of a polynomial based on an unstructured set of point and derivative values, offering a novel approach. This comparison provides a practical benchmark and highlights the strengths of the FlexHi scheme.

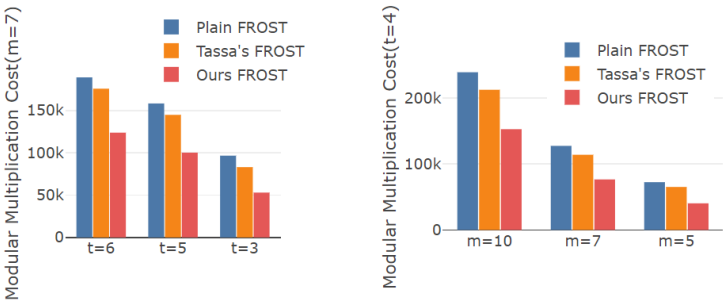

Figure 4.23: Modular Multiplication Cost of Key Generation and Key Aggregation Phases of FROST and its variants where the first bar chart represents $m = 7$ and the second bar chart represents $t = 4$

Theoretical Complexity Analysis. Table 4.3 and 4.4 analyzes the theoretical complexities of FROST protocol applications. The symbols E, M, and I in the analysis stand for modular exponentiation, modular multiplication, and modular inverse operation, respectively.

Table 4.3: Key-Related Cost Analysis of Algorithms in FROST Applications

Scheme	**KeyGen**	**KeyAgg**
Basic FROST without hierarchy $(t+1, m+1)$	$(2(m+1)(t+1) - 2(m+1) + 2)(m+1)$ E $(2(m+1)(t+1) - (t+1))(m+1)$ M $m(m+1)$ I	—
Tassa's HTSS on FROST $(\mathcal{L}_1(1,1) + \mathcal{L}_2(t,m))$	$(2mt + 2t - m + 2)(m+1)$ E $(3mt + 2t - m - 1)(m+1)$ M	—
FlexHi on FROST $(\mathcal{L}_1(1,1) + \mathcal{L}_2(t,m))$	$(2mt - 2m + 2)m + 1$ E $(2mt - t)m$ M $(m-1)m$ I	$(m+1)$ E $(2m+2)$ M m I

Table 4.4: Signature-Related Cost Analysis of Algorithms in FROST Applications

Scheme	**SignGen**	**SignAgg**
Basic FROST without hierarchy $(t+1, m+1)$	$(t+1)^2 + 3(t+1)$ E $(2(t+1)+2)(t+1) + 4(t+1) - 1$ M	—
Tassa's HTSS on FROST $(\mathcal{L}_1(1,1) + \mathcal{L}_2(t,m))$	$(t+1)^2 + 3(t+1)$ E $(2(t+1)+2)(t+1) + 4(t+1) - 1$ M	—
FlexHi on FROST $(\mathcal{L}_1(1,1) + \mathcal{L}_2(t,m))$	$(t+1)^2$ E $(2(t+1)+2)(t+1)$ M	$3t+3$ E $3t+2$ M

Since modular exponentiation and modular inverse operations can be written in terms of modular multiplication, the computation cost of these schemes can finally be examined in terms of modular multiplications. As stated in [71], modular exponentiation E takes 240 times longer than modular multiplication M, and modular inverse I requires 3 modular multiplication M. Using this information, Figure 4.23 shows the total modular multiplication cost for key generation and aggregation algorithms.

According to Figure 4.23, the first bar chart shows the proposed FlexHi FROST application runs 29 percent faster than Tassa's FROST application in the $(t, m) = (6, 7)$ case, and 36 percent faster in the $(t, m) = (3, 7)$ case. Also, the second bar chart

Table 4.5: The calculated times for different threshold values, and the number of participants

Scheme (Curve)	Threshold (t)	Time (ms)	Threshold (t)	Time (ms)
Basic FROST (ed25519)	$\mathcal{L}(4,5)$	10.104	$\mathcal{L}(3,7)$	13.453
Tassa's FROST (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,4)$	8.996	$\mathcal{L}_1(1,1)+\mathcal{L}_2(2,6)$	12.213
FlexHi FROST (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,4)$	6.498	$\mathcal{L}_1(1,1)+\mathcal{L}_2(2,6)$	10.781
Basic FROST (ed25519)	$\mathcal{L}(4,7)$	17.139	$\mathcal{L}(5,7)$	19.147
Tassa's FROST (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,6)$	15.952	$\mathcal{L}_1(1,1)+\mathcal{L}_2(4,6)$	17.829
FlexHi (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,6)$	13.280	$\mathcal{L}_1(1,1)+\mathcal{L}_2(4,6)$	15.127
Basic FROST (ed25519)	$\mathcal{L}(4,10)$	32.196	$\mathcal{L}(6,7)$	22.288
Tassa's FROST (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,9)$	29.006	$\mathcal{L}_1(1,1)+\mathcal{L}_2(5,6)$	20.624
FlexHi FROST (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,9)$	25.768	$\mathcal{L}_1(1,1)+\mathcal{L}_2(5,6)$	17.584

shows that the proposed FlexHi FROST application runs 28 percent faster than Tassa's FROST application in the $(t,m)=(4,10)$ case, and 38 percent faster in the $(t,m)=(4,5)$ case. It is clear from this that the proposed scheme works faster than Tassa's scheme.

Implementation. To confirm the aforementioned claim about theoretical efficiency, one can found the FlexHi scheme as an open source code of FROST in `https://github.com/midmotor/hierarchical-threshold-signature`. The study specifically focused on doing a thorough comparative investigation of three distinct variations of FROST. The primary objective was to evaluate the computational efficiency and applicability for a range of threshold values and participant numbers based on the theoretical framework.

Table 4.5 displays the calculated times for different parameters. In this table basic FROST corresponds to the original FROST [72] Scheme, Tassa's FROST refers to its application on FROST scheme, and FlexHi FROST refers to its application on FROST scheme. The results of the tests are performed on a computer with an i7-1165g7 @ 2.80 GHz and 16 GB RAM. The test environment incorporated two distinct elliptic curves, P256 [111] and ed25519 [21]. In Table 4.5, only results with the ed25519 curve are available; tests with the P256 curve are available in the given GitHub codes.

4.2.5 Summary

The proposed FlexHi scheme represents a significant step forward in the realm of secure digital communication and decentralized systems. Traditional threshold signature schemes, while essential, have limitations when applied to hierarchical struc-

tures, where varying levels of authority and access control are required. The proposed FlexHi scheme, built upon Shamir's construction but enhanced with independent polynomials at each hierarchical level, offers a novel and adaptable solution. It eliminates rigid constraints on threshold values, enabling the scheme to conform to a variety of real-world scenarios. Therefore, the FlexHi scheme is capable of seamlessly adapting to a wide range of real-world scenarios and organizational structures. We demonstrated its applicability by implementing it on the state-of-the-art, round-optimized FROST scheme. Considering the conducted experiments, our FROST-based FlexHi application significantly outperforms Tassa's FROST application. Based on this analysis, FlexHi scheme runs about 30% - 40% faster depending on the number of participants and the threshold values. FlexHi scheme not only introduces a more adaptable approach to hierarchical threshold signature but also demonstrates its practical advantages through faster execution.

CHAPTER 5

CONCLUSION

With the advent of the internet, voting systems transitioned to I-voting systems. In light of these systems, Helios is a pioneer for verifiable elections. While Helios offers numerous advantages to its users, it necessitates placing trust in a single server for data storage. To overcome data loss/change problems in centralized Helios server, the blockchain-powered Helios suggested using decentralized, transparent, and immutable building blocks of blockchain technology. Nevertheless, blockchain-powered Helios introduces new weaknesses related to the integration of blockchain into the voting system, such as misbehavior in wallet authorization that affects robustness, linkability in wallet authorization that affects privacy, and high cost in the transaction that affects accessibility. In this thesis, these weaknesses are successfully addressed, and the system is redesigned as Proba. In Proba, voters access the system without sacrificing privacy and robustness using the secure and efficient cryptographic threshold-issuance anonymous credential protocol. Also, a consortium blockchain made the system accessible. Briefly, Proba introduces robustness, privacy, and accessibility to blockchain-powered Helios. The critical security requirements of Proba are formally proven secure through game-based proofs. The initial considerations revolve around privacy and robustness, and the integration of the threshold-issuance anonymous credential scheme extends to include eligibility and verifiability requirements. Overall, the performance evaluation indicates that implementing the threshold-issuance anonymous credential protocol doesn't impose significant costs on the system; instead, it reduces the storage cost of smart contracts.

In addition to the design of Proba, this thesis introduces two efficient signature pro-

tocols on blockchain transactions for safeguarding the secret key of a wallet: the two-party ECDSA protocol and a flexible hierarchical threshold signature protocol. Firstly, the two-party ECDSA protocol, which has a wide range of applications in blockchain, has been improved. Many schemes have been proposed in the literature to improve the efficiency of multiparty ECDSA. Most of these schemes either require heavy homomorphic encryption computation or multiple executions of functionality that transform Multiplicative shares into Additive shares (MtA). Xue et al. [151] (CCS 2021) proposed a 2-party ECDSA protocol that is secure against malicious adversaries and only calls for one execution of MtA. The online phase only entails one party sending one field element to the other party, with the verification step of the signature scheme accounting for the majority of the computational overhead. This thesis proposes a new protocol that has the same security guarantees and the same performance in the online phase as [151]. However, it has better performance in the offline phase by lowering the computational cost by one elliptic curve multiplication and the communication cost by two field elements. The proposed protocol offers the most efficient offline phase for a two-party ECDSA protocol with such an efficient online phase. This improved protocol can be used for wallet key protection in general blockchain-based I-voting systems, thereby in Proba. Secondly, a novel, efficient, and flexible hierarchical threshold signature scheme (FlexHi) is proposed to protect the wallet secret key. This scheme is based on Shamir's architecture but enhanced with independent polynomials at each hierarchical level. Utilizing independent polynomials at each level allows for adaptability to accommodate any number of users and set threshold values without limitations. This adaptability enables us to tailor the scheme to diverse requirements, whether signing requires only top-level nodes or also lower-level participants' involvement. According to the analysis, FlexHi FROST scheme outperforms Tassa's hierarchical scheme on FROST and operates approximately 30% to 40% faster, depending on the number of participants and the chosen threshold values. This demonstrates that the FlexHi scheme has practical advantages by enhancing performance in addition to its flexibility. This novel scheme can be used to protect voters' wallet signing keys in the Proba voting system, as in the two-party ECDSA protocol, but in a hierarchical structure.

REFERENCES

[1] Y. Abuidris, R. Kumar, and W. Wenyong, A survey of blockchain based on e-voting systems, in *Proceedings of the 2019 2nd International Conference on Blockchain Technology and Applications*, pp. 99–104, 2019.

[2] B. Adida, Helios: Web-based open-audit voting., in *USENIX security symposium*, volume 17, pp. 335–348, 2008.

[3] B. Adida, O. De Marneffe, O. Pereira, J.-J. Quisquater, et al., Electing a university president using open-audit voting: Analysis of real-world use of helios, EVT/WOTE, 9(10), 2009.

[4] C. K. Adiputra, R. Hjort, and H. Sato, A proposal of blockchain-based electronic voting system, in *2018 second world conference on smart trends in systems, security and sustainability (WorldS4)*, pp. 22–27, IEEE, 2018.

[5] A. R. Ağırtaş and O. Yayla, Compartment-based and hierarchical threshold delegated verifiable accountable subgroup multi-signatures, Cryptology ePrint Archive, 2023.

[6] A. Al-Ameen and S. A. Talab, The technical feasibility and security of e-voting., Int. Arab J. Inf. Technol., 10(4), pp. 397–404, 2013.

[7] S. Al-Maaitah, M. Qatawneh, and A. Quzmar, E-voting system based on blockchain technology: A survey, in *2021 International Conference on Information Technology (ICIT)*, pp. 200–205, IEEE, 2021.

[8] M. Alharby and A. Van Moorsel, Blockchain-based smart contracts: A systematic mapping study, arXiv preprint arXiv:1710.06372, 2017.

[9] S. T. Ali and J. Murray, An overview of end-to-end verifiable voting systems, Real-World Electronic Voting, pp. 189–234, 2016.

[10] S. T. Alvi, L. Islam, T. Y. Rashme, and M. N. Uddin, Bsevoting: A conceptual framework to develop electronic voting system using sidechain, in *2021 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 10–15, IEEE, 2021.

[11] I. Bashir, *Mastering blockchain*, pp.56, Packt Publishing Ltd, 2017.

[12] A. Beimel, Secret-sharing schemes: A survey, in *International conference on coding and cryptology*, pp. 11–46, Springer, 2011.

[13] A. Beimel, T. Tassa, and E. Weinreb, Characterizing ideal weighted threshold secret sharing, in *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pp. 600–619, Springer, 2005.

[14] M. Belenkiy, Disjunctive multi-level secret sharing, Cryptology ePrint Archive, 2008.

[15] M. Bellare, A. Boldyreva, and J. Staddon, Randomness re-use in multi-recipient encryption schemeas, in *Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6*, pp. 85–99, Springer, 2002.

[16] J. Benaloh, Simple verifiable elections., EVT, 6, pp. 5–5, 2006.

[17] J. Benaloh and J. Leichter, *Generalized secret sharing and monotone functions*, Springer, 1990.

[18] J. Benaloh, R. Rivest, P. Y. Ryan, P. Stark, V. Teague, and P. Vora, End-to-end verifiability, arXiv preprint arXiv:1504.03778, 2015.

[19] J. Benaloh and D. Tuinstra, Receipt-free secret-ballot elections, in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 544–553, 1994.

[20] J. Benet, Ipfs-content addressed, versioned, p2p file system (draft 3), arXiv preprint arXiv:1407.3561, 2014.

[21] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, High-speed high-security signatures, Journal of cryptographic engineering, 2(2), pp. 77–89, 2012.

[22] R. Bhardwaj and D. Datta, *Consensus Algorithm*, pp. 91–107, Springer International Publishing, Cham, 2020, ISBN 978-3-030-38677-1.

[23] P. Bibiloni, A. Escala, and P. Morillo, Vote validatability in mix-net-based evoting, in *International Conference on E-Voting and Identity*, pp. 92–109, Springer, 2015.

[24] G. R. Blakley, Safeguarding cryptographic keys, in *Managing Requirements Knowledge, International Workshop on*, pp. 313–313, IEEE Computer Society, 1979.

[25] D. Boneh, B. Lynn, and H. Shacham, Short signatures from the Weil pairing, in *International conference on the theory and application of cryptology and information security*, pp. 514–532, Springer, 2001.

[26] D. Boneh and V. Shoup, A graduate course in applied cryptography, Stanford University (Version 0.5), 2020.

[27] E. F. Brickell, Some ideal secret sharing schemes, in *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 468–475, Springer, 1989.

[28] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, UC non-interactive, proactive, threshold ECDSA with identifiable aborts, IACR Cryptol. ePrint Arch., p. 60, 2021.

[29] A. S. Cardozo and Z. Williamson, Eip 1108: Reduce alt_bn128 precompile gas costs, Ethereum Improvement Proposals, (1108), 2018.

[30] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, Two-party ecdsa from hash proof systems and efficient instantiations, in A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pp. 191–221, Springer International Publishing, Cham, 2019, ISBN 978-3-030-26954-8.

[31] O. Cetinkaya and D. Cetinkaya, Towards secure e-elections in turkey: requirements and principles, in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pp. 903–907, IEEE, 2007.

[32] D. Chaum, Blind signatures for untraceable payments, in *Advances in cryptology*, pp. 199–203, Springer, 1983.

[33] D. Chaum and T. P. Pedersen, Wallet databases with observers, in *Annual international cryptology conference*, pp. 89–105, Springer, 1992.

[34] D. L. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, Communications of the ACM, 24(2), pp. 84–90, 1981.

[35] B. Chevallier-Mames, P.-A. Fouque, D. Pointcheval, J. Stern, and J. Traoré, On some incompatible properties of voting schemes, in *Towards Trustworthy Elections*, pp. 191–199, Springer, 2010.

[36] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, A homomorphic lwe based e-voting scheme, in *Post-Quantum Cryptography*, pp. 245–265, Springer, 2016.

[37] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, Distributed elgamal á la pedersen: application to helios, in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, pp. 131–142, 2013.

[38] V. Cortier, P. Gaudry, and S. Glondu, Belenios: a simple private and verifiable electronic voting system, in *Foundations of Security, Protocols, and Equational Reasoning*, pp. 214–238, Springer, 2019.

[39] N. T. Courtois, P. Emirdag, and F. Valsorda, Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor rng events, Cryptology ePrint Archive, 2014.

[40] R. Cramer, R. Gennaro, and B. Schoenmakers, A secure and optimally efficient multi-authority election scheme, European transactions on Telecommunications, 8(5), pp. 481–490, 1997.

[41] E. Daniel and F. Tschorsch, Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks, IEEE Communications Surveys & Tutorials, 24(1), pp. 31–52, 2022.

[42] Y. Desmedt, Society and group oriented cryptography: a new concept, in C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pp. 120–127, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, ISBN 978-3-540-48184-3.

[43] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, Secure two-party threshold ecdsa from ecdsa assumptions, in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 980–997, IEEE, 2018.

[44] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, Threshold ECDSA from ECDSA assumptions: The multiparty case, 2019 IEEE Symposium on Security and Privacy (SP), pp. 1051–1066, 2019.

[45] R. Dutta, R. Barua, and P. Sarkar, Pairing-based cryptographic protocols: A survey, Cryptology ePrint Archive, 2004.

[46] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE transactions on information theory, 31(4), pp. 469–472, 1985.

[47] O. Ersoy, K. Kaya, and K. Kaskaloglu, Multilevel threshold secret and function sharing based on the chinese remainder theorem, Preprint arXiv:1605.07988, 2016.

[48] A. Fiat and A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, in *Conference on the theory and application of cryptographic techniques*, pp. 186–194, Springer, 1986.

[49] A. Fujioka, T. Okamoto, and K. Ohta, A practical secret voting scheme for large scale elections, in *International Workshop on the Theory and Application of Cryptographic Techniques*, pp. 244–251, Springer, 1992.

[50] A. J. Gabriel, B. K. Alese, A. O. Adetunmbi, O. S. Adewale, and O. A. Sarumi, Post-quantum crystography system for secure electronic voting, Open Computer Science, 9(1), pp. 292–298, 2019.

[51] D. Gaweł, M. Kosarzecki, P. L. Vora, H. Wu, and F. Zagórski, Apollo–end-to-end verifiable internet voting with recovery from vote manipulation, in *International Joint Conference on Electronic Voting*, pp. 125–143, Springer, 2017.

[52] R. Gennaro and S. Goldfeder, Fast multiparty threshold ecdsa with fast trustless setup, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1179–1194, 2018.

[53] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, Secure distributed key generation for discrete-log based cryptosystems, in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 295–310, Springer, 1999.

[54] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, Secure distributed key generation for discrete-log based cryptosystems, Journal of Cryptology, 20, pp. 51–83, 2007.

[55] H. Ghodosi, J. Pieprzyk, and R. Safavi-Naini, Secret sharing in multilevel and compartmented groups, in *Information Security and Privacy: ACISP'98 Brisbane, Australia*, pp. 367–378, Springer, 1998.

[56] S. Goldwasser and S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, p. 365–377, Association for Computing Machinery, New York, NY, USA, 1982, ISBN 0897910702.

[57] S. Goldwasser and S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 365–377, 1982.

[58] S. Haber, J. Benaloh, and S. Halevi, The helios e-voting demo for the iacr, International association for cryptologic research, 2010.

[59] L. Harn and M. Fuyou, Multilevel threshold secret sharing based on the chinese remainder theorem, Information processing letters, 114(9), pp. 504–509, 2014.

[60] M. P. Heinl, S. Gölz, and C. Bösch, Remote electronic voting in uncontrolled environments: A classifying survey, ACM Computing Surveys, 55(8), pp. 1–44, 2022.

[61] J. Huang, D. He, M. S. Obaidat, P. Vijayakumar, M. Luo, and K.-K. R. Choo, The application of the blockchain technology in voting systems: A review, ACM Computing Surveys (CSUR), 54(3), pp. 1–28, 2021.

[62] U. Jafar, M. J. A. Aziz, and Z. Shukur, Blockchain for electronic voting system-review and open research challenges, Aug 2021.

[63] D. Jefferson, D. Buell, K. Skoglund, J. Kiniry, and J. Greenbaum, What we don't know about the voatz "blockchain" internet voting system, University of South Carolina, South Carolina, 2019.

[64] R. Joaquim, How to prove the validity of a complex ballot encryption to the voter and the public, Journal of information security and applications, 19(2), pp. 130–142, 2014.

[65] H. Jonker and J. Pang, Bulletin boards in voting systems: Modelling and measuring privacy, in *2011 Sixth International Conference on Availability, Reliability and Security*, pp. 294–300, IEEE, 2011.

[66] A. Juels, D. Catalano, and M. Jakobsson, Coercion-resistant electronic elections, in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, p. 61–70, Association for Computing Machinery, New York, NY, USA, 2005, ISBN 1595932283.

[67] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer, Usability analysis of helios—an open source verifiable remote electronic voting system, in *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*, 2011.

[68] E. Käsper, V. Nikov, and S. Nikova, Strongly multiplicative hierarchical threshold secret sharing, in *Information Theoretic Security: Second International Conference, ICITS 2007, Madrid, Spain, May 25-29, 2007, Revised Selected Papers 2*, pp. 148–168, Springer, 2009.

[69] Y.-X. Kho, S.-H. Heng, and J.-J. Chin, A review of cryptographic electronic voting, Symmetry, 14(5), p. 858, 2022.

[70] C. Killer, B. Rodrigues, R. Matile, E. Scheid, and B. Stiller, Design and implementation of cast-as-intended verifiability for a blockchain-based voting system, in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 286–293, 2020.

[71] N. Koblitz, A. Menezes, and S. Vanstone, The state of elliptic curve cryptography, Designs, codes and cryptography, 19, pp. 173–193, 2000.

[72] C. Komlo and I. Goldberg, FROST: flexible round-optimized schnorr threshold signatures, in *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, pp. 34–65, Springer, 2021.

[73] D. W. Kravitz, Digital signature algorithm, July 27 1993, uS Patent 5,231,668.

[74] R. Krimmer, S. Triessnig, and M. Volkamer, The development of remote e-voting around the world: A review of roads and directions, in *E-Voting and Identity: First International Conference, VOTE-ID 2007, Bochum, Germany, October 4-5, 2007, Revised Selected Papers 1*, pp. 1–15, Springer, 2007.

[75] M. Kumar, C. P. Katti, and P. C. Saxena, A secure anonymous e-voting system using identity-based blind signature scheme, in *International conference on information systems security*, pp. 29–49, Springer, 2017.

[76] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, Ordinos: A verifiable tally-hiding e-voting system, in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 216–235, IEEE, 2020.

[77] P. Labs, Filecoin: A decentralized storage network, https://research.protocol.ai/publications/filecoin-a-decentralized-storage-network/protocollabs2017a.pdf, (Accessed:10/10/2022).

[78] P. Lam, From helios to voatz: Blockchain voting and the vulnerabilities it opens for the future, https://www.cs.tufts.edu/comp/116/archive/fall2019/plam.pdf, (Accessed: 13/11/2022).

[79] K. Lee, J. I. James, T. G. Ejeta, and H. J. Kim, Electronic voting service using block-chain, Journal of Digital Forensics, Security and Law, 11(2), p. 8, 2016.

[80] L. Li, An electronic voting scheme based on elgamal homomorphic encryption for privacy protection, in *Journal of Physics: Conference Series*, volume 1544, p. 012036, IOP Publishing, 2020.

[81] Y. Lindell, Fast secure two-party ECDSA signing, in J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pp. 613–644, Springer, 2017.

[82] Y. Lindell, Secure multiparty computation, Communications of the ACM, 64(1), pp. 86–96, 2020.

[83] Y. Lindell and A. Nof, Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1837–1854, 2018.

[84] H. Lipmaa and T. Toft, Secure equality and greater-than tests with sublinear online complexity, in *International Colloquium on Automata, Languages, and Programming*, pp. 645–656, Springer, 2013.

[85] E. Maaten, Towards remote e-voting: Estonian case, in *Electronic voting in Europe-Technology, law, politics and society, workshop of the ESF TED programme together with GI and OCG*, Gesellschaft für Informatik eV, 2004.

[86] P. MacKenzie and M. K. Reiter, Two-party generation of dsa signatures, in *Annual International Cryptology Conference*, pp. 137–154, Springer, 2001.

[87] E. Magkos, P. Kotzanikolaou, and C. Douligeris, Towards secure online elections: models, primitives and open issues, Electronic Government, 4(3), pp. 249–268, 2007.

[88] V. Mateu, S. Martínez, J. M. Miret, F. Sebé, et al., Elliptic curve array ballots for homomorphic tallying elections, in *International Conference on Electronic Government and the Information Systems Perspective*, pp. 334–347, Springer, 2015.

[89] E. McMurtry, X. Boyen, C. Culnane, K. Gjøsteen, T. Haines, and V. Teague, Towards verifiable remote voting with paper assurance, arXiv preprint arXiv:2111.04210, 2021.

[90] A. Menezes, U. of Waterloo. Dept. of Combinatorics, Optimization, and U. of Waterloo. Faculty of Mathematics, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Faculty of Mathematics, University of Waterloo, 1999.

[91] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, A review on consensus algorithm of blockchain, in *2017 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 2567–2572, IEEE, 2017.

[92] S. Mitsunari, R. Sakai, and M. Kasahara, A new traitor tracing, IEICE transactions on fundamentals of electronics, communications and computer sciences, 85(2), pp. 481–484, 2002.

[93] H. Moniz, The istanbul bft consensus algorithm, arXiv preprint arXiv:2002.03613, 2020.

[94] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Decentralized Business Review, p. 21260, 2008.

[95] S. Nakamoto and A. Bitcoin, A peer-to-peer electronic cash system, Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf, 4, p. 2, 2008.

[96] M. Niranjanamurthy, B. Nithya, and S. Jagannatha, Analysis of blockchain technology: pros, cons and swot, Cluster Computing, 22, pp. 14743–14757, 2019.

[97] M. Nojoumian and D. R. Stinson, Sequential secret sharing as a new hierarchical access structure, Cryptology ePrint Archive, 2015.

[98] O. Okediran, E. Omidiora, S. Olabiyisi, and R. Ganiyu, A comparative study of generic cryptographic models for secure electronic voting, British Journals of Science, 1(2), pp. 135–142, 2011.

[99] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in *International conference on the theory and applications of cryptographic techniques*, pp. 223–238, Springer, 1999.

[100] N. Pakniat, M. Noroozi, and Z. Eslami, Distributed key generation protocol with hierarchical threshold access structure, IET Information Security, 9(4), pp. 248–255, 2015.

[101] T. P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in *Annual international cryptology conference*, pp. 129–140, Springer, 1991.

[102] T. P. Pedersen, A threshold cryptosystem without a trusted party, in *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pp. 522–526, Springer, 1991.

[103] L. Peng, W. Feng, Z. Yan, Y. Li, X. Zhou, and S. Shimizu, Privacy preservation in permissionless blockchain: A survey, Digital Communications and Networks, 7(3), pp. 295–307, 2021.

[104] B. M. B. Pereira, J. M. Torres, P. M. Sobral, R. S. Moreira, C. P. d. A. Soares, and I. Pereira, Blockchain-based electronic voting: A secure and transparent solution, Cryptography, 7(2), p. 27, 2023.

[105] O. Pereira, Internet voting with helios, Real-World Electronic Voting: Design, Analysis and Deployment, 8604, 2016.

[106] A. J. Perez and E. N. Ceesay, Improving end-to-end verifiable voting systems with blockchain technologies, in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1108–1115, IEEE, 2018.

[107] D. Pointcheval and O. Sanders, Short randomizable signatures, in *Cryptographers' Track at the RSA Conference*, pp. 111–126, Springer, 2016.

[108] D. Pointcheval and J. Stern, Security arguments for digital signatures and blind signatures, Journal of cryptology, 13, pp. 361–396, 2000.

[109] T. Pornin, Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa), Technical report, 2013.

[110] J. Puiggalí-Allepuz and S. Guasch-Castelló, Privacy and anonymity management in electronic voting, Serbian Publication InfoReview joins UPENET, the Network of CEPIS Societies Journals and Magazines, p. 59, 2010.

[111] M. Qu, Sec 2: Recommended elliptic curve domain parameters, Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6, 1999.

[112] E. A. Quaglia and B. Smyth, A short introduction to secrecy and verifiability for elections, arXiv preprint arXiv:1702.03168, 2017.

[113] A. Rial and A. M. Piotrowska, Security analysis of coconut, an attribute-based credential scheme with threshold issuance, Cryptology ePrint Archive, 2022.

[114] P. Ribarski and L. Antovski, Introduction to secure electronic voting requirements, in *The 7th International Conference for Informatics and Information Technology*, Institute of Informatics, Faculty of Natural Sciences and Mathematics, 2010.

[115] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, 21(2), pp. 120–126, 1978.

[116] S. S. Sabry, N. M. Kaittan, and I. Majeed, The road to the blockchain technology: Concept and types, Periodicals of Engineering and Natural Sciences, 7(4), pp. 1821–1832, 2019.

[117] R. Saltini and D. Hyland-Wood, Correctness analysis of ibft, arXiv preprint arXiv:1901.07160, 2019.

[118] K. Sampigethaya and R. Poovendran, A framework and taxonomy for comparison of electronic voting schemes, computers & security, 25(2), pp. 137–153, 2006.

[119] A. Schneider, C. Meter, and P. Hagemeister, Survey on remote electronic voting, arXiv preprint arXiv:1702.02798, 2017.

[120] N. Schneider, Recovering bitcoin private keys using weak signatures from the blockchain, `https://web.archive.org/web/20160308014317/http://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html`, Accessed: (27.01.2024).

[121] C. Schnorr, Efficient signature generation by smart cards, in *Advances in Cryptology — CRYPTO '87*, pp. 161–174, 1991.

[122] C. P. Schnorr, Efficient identification and signatures for smart cards, in G. Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pp. 239–252, Springer New York, New York, NY, 1990.

[123] T. C. U. Senate, Voatz,2017, https://voatz.com/, (Accessed: 01/11/2022).

[124] A. Shamir, How to share a secret, Commun. ACM, 22(11), p. 612–613, nov 1979, ISSN 0001-0782.

[125] F. Shirazi, S. Neumann, I. Ciolacu, and M. Volkamer, Robust electronic voting: Introducing robustness in civitas, in *2011 International Workshop on Requirements Engineering for Electronic Voting Systems*, pp. 47–55, IEEE, 2011.

[126] P. W. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[127] V. Shoup, Sequences of games: a tool for taming complexity in security proofs, cryptology eprint archive, 2004.

[128] G. J. Simmons, How to (really) share a secret, in *Conference on the Theory and Application of Cryptography*, pp. 390–448, Springer, 1988.

[129] A. Sonnino, Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers, 2019.

[130] G. Srivastava, S. Dhar, A. D. Dwivedi, and J. Crichigno, Blockchain education, in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–5, IEEE, 2019.

[131] C. Staff, Acm's 2014 general election: please take this opportunity to vote, Communications of the ACM, 57(5), pp. 9–17, 2014.

[132] D. R. Stinson, An explication of secret sharing schemes, Designs, Codes and Cryptography, 2(4), pp. 357–390, 1992.

[133] S. Suratkar, M. Shirole, and S. Bhirud, Cryptocurrency wallet: A review, in *2020 4th international conference on computer, communication and signal processing (ICCCSP)*, pp. 1–7, IEEE, 2020.

[134] N. Szabo, The idea of smart contracts, Nick Szabo's papers and concise tutorials, 6(1), p. 199, 1997.

[135] R. Taş and Ö. Ö. Tanrıöver, A systematic review of challenges and opportunities of blockchain for e-voting, Symmetry, 12(8), p. 1328, 2020.

[136] T. Tassa, Hierarchical threshold secret sharing, in *Theory of Cryptography Conference*, pp. 473–490, Springer, 2004.

[137] T. Tassa and N. Dyn, Multipartite secret sharing by bivariate interpolation, Journal of Cryptology, 22, pp. 227–258, 2009.

[138] A. N. Tentu, P. Paul, and V. C. Venkaiah, Ideal and perfect hierarchical secret sharing schemes based on mds codes, Cryptology ePrint Archive, 2013.

[139] A. T. N. Thi and T. K. Dang, Privacy preserving in electronic voting, Electrical and Electronics Engineering Computer and Information Engineering, p. 28, 2014.

[140] G. Tillem, O. Burundukov, and I. Team, Threshold signatures using secure multiparty computation, `https://www.ingwb.com/binaries/content/assets/insights/themes/distributed-ledger-technology/ing-releases-multiparty-threshold-signing-library-to-improve-customer-security/threshold-signatures-using-secure-multiparty-computation.pdf`, Accessed: (25.05.2021).

[141] G. Traverso, D. Demirel, and J. Buchmann, Performing computations on hierarchically shared secrets, in *Progress in Cryptology–AFRICACRYPT 2018: 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10*, pp. 141–161, Springer, 2018.

[142] E. Tsekmezoglou and J. Iliadis, A critical view on internet voting technology, The Electronic Journal for E-Commerce Tools and Applications. Disponibile su http://minbar. cs. dartmouth. edu/greecom/ejeta/fourth, (2005), 2005.

[143] G. Tsoukalas, K. Papadimitriou, and P. Louridas, From helios to zeus, USENIX Journal of Election Technology and Systems (JETS), 1(1), pp. 1–17, 2013.

[144] P. University, Helios princeton undergraduate elections, https://princeton.heliosvoting.org/, (Accessed: 01/05/2022).

[145] J. L. Villar, Zero-knowledge proofs notes, `https://web.mat.upc.edu/jorge.villar/doc/notes/DataProt/zk.html#section2.2.3`, Accessed: (15.01.2021).

[146] M.-V. Vladucu, Z. Dong, J. Medina, and R. Rojas-Cessa, E-voting meets blockchain: A survey, IEEE Access, 11, pp. 23293–23308, 2023.

[147] K.-H. Wang, S. K. Mondal, K. Chan, and X. Xie, A review of contemporary e-voting: Requirements, technology, systems and usability, Data Science and Pattern Recognition, 1(1), pp. 31–47, 2017.

[148] A. Wigderson, O. Goldreich, and S. Micali, How to play any mental game [c], in *Procedings the 19th Annual ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[149] G. Wood et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper, 151(2014), pp. 1–32, 2014.

[150] Z. Xia, Z. Yang, S. Xiong, and C.-F. Hsu, Game-based security proofs for secret sharing schemes, in *Second International Conference on Security with Intelligent Computing and Big Data Services (SICBS-2018)*, pp. 650–660, Springer, 2020.

[151] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, Efficient online-friendly two-party ecdsa signature, in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 558–573, 2021.

[152] A. C. Yao, Protocols for secure computations, in *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164, IEEE, 1982.

[153] R. Yılmaz, *Some ideal secret sharing schemes*, Ph.D. thesis, Bilkent Universitesi (Turkey), 2010.

[154] J. Yuan, J. Yang, C. Wang, X. Jia, F.-W. Fu, and G. Xu, A new efficient hierarchical multi-secret sharing scheme based on linear homogeneous recurrence relations, Information Sciences, 592, pp. 36–49, 2022.

[155] S. Zhang and J.-H. Lee, Analysis of the main consensus protocols of blockchain, ICT express, 6(2), pp. 93–97, 2020.

[156] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, An overview of blockchain technology: Architecture, consensus, and future trends, 2017 IEEE International Congress on Big Data (BigData Congress), pp. 557–564, 2017.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Kocaman, Sermin
Nationality: Turkish (TC)

EDUCATION

Degree	Institution	Year of Graduation
B.S.	Department Of Mathematics, METU	2019

PUBLICATIONS

- Kocaman, S., & Talibi Alaoui, Y. (2023, December). Efficient Secure Two Party ECDSA. In IMA International Conference on Cryptography and Coding (pp. 161-180). Cham: Springer Nature Switzerland.

- Bingol, M.A., Kocaman, S., Dogan, A., & Kurt Toplu, S. FlexHi: A Flexible Hierarchical Threshold Signature Scheme. In Computing Conference 2024. https://eprint.iacr.org/2024/024

- Doröz, Y., Kocaman, S., Muş, K., Sulak, F., & Yayla, O. Proba: Privacy-preserving, Robust and Accessible Blockchain-powered Helios. Journal of Information Security and Applications (Submitted)