Distributed Cryptographic Protocols

PhD Thesis

Antonio Manuel Larriba Flor

Supervised by

Dr. D. Damián López Rodríguez
Dr. D. José María Sempere Luna

May 9, 2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Agradecimientos

Ha llegado la hora de escribir estas líneas, quizás un poco después de lo que tenía planeado y con algún año más. El viaje no ha sido siempre fácil, pero desde luego ha sido entretenido. Y si algo ha hecho más llevadero estos años ha sido la ayuda recibida, y yo tengo mucho que agradecer.
En primer lugar, si he de agradecer a alguien es a mis tutores, por su ayuda y su guía a lo largo de este proceso. Especialmente a Damián, por transmitirme su pasión por la criptografía en la carrera, su confianza y ayudarme cuando más perdido estaba en mi camino como estudiante. También quiero dar las gracias a Chema por su apoyo, las revisiones y por hacerme entender que hay problemas que no tienen solución.

También quiero agradecer a mis amigos el apoyo que me han dado todos estos años, a los de siempre y a los que conocí en el camino. Por las charlas, los consejos, las visitas a las principales vías fluviales de Europa, los buenos ratos y por poder evadirnos un rato.

A Manuela, por hacerme tan feliz, y por quererme y aguantarme incluso cuando no me lo merezco. Por estar allí siempre, por recogerme del suelo metafórica y literalmente, por las arepas, los montajes y los gatos.

Por último, a los que siempre me han ayudado y dado su cariño, a mi familia. A mi hermano por entenderme y ayudarme, a mi padre por sus consejos y sus enseñanzas, y a mi madre por su sentido común y su constancia. Sin ellos nada de esto hubiera sido posible.

Valencia. 9 de Marzo, 2023

Abstract

Trust is the base of modern societies. However, trust is difficult to achieve and can be exploited easily with devastating results. In this thesis, we explore the use of distributed cryptographic protocols to build reliable systems where trust can be replaced by cryptographic and mathematical guarantees. In these adaptive systems, even if one involved party acts dishonestly, the integrity and robustness of the system can be ensured as there exist mechanisms to overcome these scenarios. Therefore, there is a transition from systems based in trust, to schemes where trust is distributed between decentralized parties. Individual parties can be audited, and universal verifiability ensures that any user can compute the final state of these methods, without compromising individual users' privacy.

Most collaboration problems we face as societies can be reduced to two main dilemmas: voting on a proposal or electing political representatives, or identifying ourselves as valid members of a collective to access a service or resource. Hence, this doctoral thesis focuses on distributed cryptographic protocols for electronic voting and anonymous identification.

We have developed three electronic voting schemes that enhance traditional methods, and protect the privacy of electors while ensuring the integrity of the whole election. In these systems, we have employed different cryptographic mechanisms, that fulfill all the desired security properties of an electronic voting scheme, under different assumptions. Some of them are secure even in post-quantum scenarios. We have provided a detailed time-complexity analysis to prove that our proposed methods are efficient and feasible to implement. We also implemented some voting protocols, or parts of them, and carried out meticulous experimentation to show the potential of our contributions.

Finally, we study in detail the identification problem and propose three distributed and non-interactive methods for anonymous registration and access. These three protocols are especially lightweight and application agnostic, making them feasible to be integrated with many purposes. We formally analyze and demonstrate the security of our identification protocols, and provide a complete implementation of them to once again show the feasibility and effectiveness of the developed solutions. Using this identification framework, we can ensure the security of the guarded resource, while also preserving the anonymity of the users.

Resumen

La confianza es la base de las sociedades modernas. Sin embargo, las relaciones basadas en confianza son difíciles de establecer y pueden ser explotadas fácilmente con resultados devastadores. En esta tesis exploramos el uso de protocolos criptográficos distribuidos para construir sistemas confiables donde la confianza se vea reemplazada por garantías matemáticas y criptográficas. En estos nuevos sistemas dinámicos, incluso si una de las partes se comporta de manera deshonesta, la integridad y resiliencia del sistema están garantizadas, ya que existen mecanismos para superar este tipo de situaciones. Por lo tanto, hay una transición de sistemas basados en la confianza, a esquemas donde esta misma confianza es descentralizada entre un conjunto de individuos o entidades. Cada miembro de este conjunto puede ser auditado, y la verificación universal asegura que todos los usuarios puedan calcular el estado final en cada uno de estos métodos, sin comprometer la privacidad individual de los usuarios.

La mayoría de los problemas de colaboración a los que nos enfrentamos como sociedad, pueden reducirse a dos grandes dilemas: el votar una propuesta, o un representante político, ó identificarnos a nosotros mismos como miembros de un colectivo con derecho de acceso a un recurso o servicio. Por ello, esta tesis doctoral se centra en los protocolos criptográficos distribuidos aplicados al voto electrónico y la identificación anónima.

Hemos desarrollado tres protocolos para el voto electrónico que complementan y mejoran a los métodos más tradicionales, y además protegen la privacidad de los votantes al mismo tiempo que aseguran la integridad del proceso de voto. En estos sistemas, hemos empleado diferentes mecanismos criptográficos que proveen, bajo diferentes asunciones, de las propiedades de seguridad que todo sistema de voto debe tener. Algunos de estos sistemas son seguros incluso en escenarios pos-cuánticos. También hemos calculado minuciosamente la complejidad temporal de los métodos para demostrar que son eficientes y factibles de ser implementados. Además, hemos implementado algunos de estos sistemas, o partes de ellos, y llevado a cabo una detallada experimentación para demostrar el potencial de nuestras contribuciones.

Finalmente, estudiamos en detalle el problema de la identificación y proponemos tres métodos no interactivos y distribuidos que permiten el registro y acceso anónimo. Estos protocolos son especialmente ligeros y agnósticos en su implementación, lo que permite que puedan ser integrados con múltiples propósitos. Hemos formalizado y demostrado la seguridad de nuestros protocolos de identificación, y hemos realizado una implementación completa de ellos para, una vez más, demostrar la factibilidad y eficiencia de las soluciones propuestas. Bajo este marco teórico de identificación, somos capaces de asegurar el recurso custodiado, sin que ello suponga una violación para el anonimato de los usuarios.

Resum

La confiança és la base de les societats modernes. No obstant això, les relacions basades en confiança són difícils d'establir i poden ser explotades fàcilment amb resultats devastadors. En aquesta tesi explorem l'ús de protocols criptogràfics distribuïts per a construir sistemes de confiança on la confiança es veja reemplaçada per garanties matemàtiques i criptogràfiques. En aquests nous sistemes dinàmics, fins i tot si una de les parts es comporta de manera deshonesta, la integritat i resiliència del sistema estan garantides, ja que existeixen mecanismes per a superar aquest tipus de situacions. Per tant, hi ha una transició de sistemes basats en la confiança, a esquemes on aquesta acarona confiança és descentralitzada entre un conjunt d'individus o entitats. Cada membre d'aquest conjunt pot ser auditat, i la verificació universal assegura que tots els usuaris puguen calcular l'estat final en cadascun d'aquests mètodes, sense comprometre la privacitat individual dels usuaris.

La majoria dels problemes de colůlaboració als quals ens enfrontem com a societat, poden reduir-se a dos grans dilemes: el votar una proposta, o un representant polític, o identificar-nos a nosaltres mateixos com a membres d'un colůlectiu amb dret d'accés a un recurs o servei. Per això, aquesta tesi doctoral se centra en els protocols criptogràfics distribuïts aplicats al vot electrònic i la identificació anònima.

Hem desenvolupat tres protocols per al vot electrònic que complementen i milloren als mètodes més tradicionals, i a més protegeixen la privacitat dels votants al mateix temps que asseguren la integritat del procés de vot. En aquests sistemes, hem emprat diferents mecanismes criptogràfics que proveeixen, baix diferents assumpcions, de les propietats de seguretat que tot sistema de vot ha de tindre. Alguns d'aquests sistemes són segurs fins i tot en escenaris post-quàntics. També hem calculat minuciosament la complexitat temporal dels mètodes per a demostrar que són eficients i factibles de ser implementats. A més, hem implementats alguns d'aquests sistemes, o parts d'ells, i dut a terme una detallada experimentació per a demostrar la potencial de les nostres contribucions.

Finalment, estudiem detalladament el problema de la identificació i proposem tres mètodes no interactius i distribuïts que permeten el registre i accés anònim. Aquests protocols són especialment lleugers i agnòstics en la seua implementació, la qual cosa permet que puguen ser integrats amb múltiples propòsits. Hem formalitzat i demostrat la seguretat dels nostres protocols d'identificació, i hem realitzat una implementació completa d'ells per a, una vegada més, demostrar la factibilitat i eficiència de les solucions proposades. Sota aquest marc teòric d'identificació, som capaces d'assegurar el recurs custodiat, sense que això supose una violació per a l'anonimat dels usuaris.

Notation

Symbol	References to
$a\|\|b$	Concatenation.
$a \cdot b$	Scalar multiplication.
$a \cdot b \mod n$	Modular multiplication.
$a^{-1} \mod n$	Inverse modulo n.
$\odot, \oplus$	General binary operation.
$a, \ldots, z$	Scalars.
$A, \ldots, Z$	Generators, or elliptic curve points.
$\lvert \cdot \rvert$	Cardinality of a group, field, or set.
$\{\cdot, \ldots, \cdot\}$	Unordered list.
$\langle \cdot, \ldots, \cdot \rangle$	Ordered list.
$f(x)$	Polynomial.
$deg(\cdot)$	Degree of a polynomial.
G	Group.
G_p	Finite group.
F	Field.
GF	Galois Field.
F_p	Finite field.
F_{p^k}	Extended finite field.
$F[x]$	Set of all possible polynomials over a field F.
$ord(\cdot)$	Order of a group.
$E(F)$	Elliptic curve E over a finite field F.
$a \in A$.	Element in a group, field or set.
$a \in_R A$.	Element sampled at random in a group, field or set.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Reputation is what other people know about you. Honor is what you know about yourself.

Lois McMaster Bujold

Uhsxwdwlrq lv zkdw rwkhu shrsoh nqrz derxw brx. Krqru lv zkdw brx nqrz derxw brxuvhoi.

Caesar Cipher, $k = 3$

As human beings, we are social by nature. From ancient Greece, through the Roman Empire, the Middle Ages, the Renaissance, the Enlightenment, the Industrial Revolution, to the modern age, human societies are collaborative, and therefore based on trust. Social organizations and government systems have varied through history, but not the need for association and the fundamental aspect of trust in these systems. Trust enables us to interact with others and engage in various social, economic, and political activities. However, in many cases, trust is difficult to establish, and it can be challenging to ensure that individuals act honestly and transparently.

Game theory has extensively studied collaboration theory Sachs et al. [2004], Arsenyan et al. [2015], Cai and Kock [2009]. In most cooperative dilemmas, we can differentiate two outcomes: the general one, that affects the system as a whole and averages the result between the involved parties, and the individual one. And while collaboration might be the optimal solution for the general outcome of the system, individual parties might benefit, usually by worsening the outcome of other parties, if they cheat or lie. As individuals, we are also suited to do good, (e.g. empathy, sharing, helping), but most social organizations require a hierarchical structure to function properly. Hence, we have people with varying power and responsibilities in our social structures. This is not bad per se, but it introduces new challenges.

- Power tends to corrupt, and absolute power corrupts absolutely. While as individuals, we are also suited to do good, (e.g. empathy, sharing, helping), the more power, the more potential reward by deceiving the system.
- To ensure the honest cooperation of most parties, some restrictions need to be added to these social organizations. Once again, this is not bad, but the implementation of these restrictions usually degrades individual privacy in favor of a secure system.

Privacy is a fundamental human right, and it is essential for the functioning of a democratic society. Cryptography provides a solution to these challenges by allowing the creation of systems where trust is replaced by mathematical proofs, while also respecting the privacy of the individuals that operate within these systems. However, these cryptography-based systems also raise many challenges, including the need to ensure the security and reliability of the protocols, and the need to balance privacy with other values, such as transparency and accountability. Most of the collaborative challenges we face as democratic societies can be reduced to two general and related problems: voting on a specific topic or handling access to a protected or limited resource. Voting is intrinsically connected to the governance of the hierarchical structure that presides over our society and shapes the legal and economic framework within which we function. Considering its critical role in our lives and the substantial economic and power incentives for malicious actors to interfere with electoral processes, it is essential to emphasize the importance of conducting fair, transparent, and secure elections. The second issue, ensuring secure access to a resource, is equally significant in guaranteeing a fair and authenticated allocation of resources among parties. This matter is closely tied to voting, as enforcing appropriate and restricted access to a resource, such as a ballot, represents the cornerstone challenge of voting systems. However, secure access extends beyond voting and is relevant to a variety of other contexts, including online services, reservations, grant management, and determining eligibility, among others.

In this doctoral thesis, we focus on distributed cryptographic protocols that enable electronic voting and anonymous identification. Our goal is the research and implementation of secure and reliable protocols that can be used in real-world scenarios. We believe that our work can contribute to the development of secure and privacy-preserving electronic voting and identification systems that can be used in a democratic society. We hope that our research will enable individuals and groups to act collectively in an honest way, without sacrificing their privacy, and will contribute to the protection of fundamental human rights in the digital age.

1.1 Distributed Cryptography

Along this thesis, we employ the term Distributed Cryptography, or distributed cryptographic protocols, to refer to any kind of cryptographic primitive, or protocol, that is executed along a distributed network, to disseminate power and redistribute responsibility among a set of parties, as opposed to a centralized computation.

This definition includes, but is broader than, some cryptography terms such as:

Secure multi-party computation Cramer et al. [2015], Canetti et al. [1996], where a set of parties collaborate to jointly compute a function over some secret inputs. In this framework, parties want to preserve their secret inputs from other parties.

Distributed key generation protocols Damgård and Koprowski [2001], Frankel et al. [1998] where a set of parties contribute to the calculation of a shared public key component, and each of them ends with a private share of the private key component.

Threshold systems Agrawal et al. [2018], Rabin [1998] where at least a minimum set of parties, of a limited set of them, need to collaborate to be able to jointly compute a function.

Secret Sharing schemes (see Section 2.2), where a secret is shared in multiple pieces that do not reveal any information about the secret itself, between a set of parties. Usually, these parties collaborate under a threshold scheme to recover the original secret.

All these terms are tightly coupled, and many times secure multi-party computation is used as a general term, while other schemes are considered methods within the framework, and secret sharing schemes are treated as the basic building block for those systems. However, this same blend also causes many specifics of these different schemes to get lost, as sometimes they are used indistinctly.

Therefore, to mitigate this ambiguity, and refer only to protocols that actually use this framework to achieve a real decentralization of power, and not only a centralized system with improved reliability.

1.2 Electronic Voting

Voting is an essential process in any democratic society. It allows individuals to express their preferences and elect representatives who will act on their behalf. However, voting is not a simple process, and it involves many challenges. Traditional voting systems usually rely on paper ballots, and the collaboration of individuals from the whole political spectrum to address these issues. As the more people with different ideologies oversee the voting process, the more secure the election is. This idea of distributing the trust between as many varied parties as possible is powerful, but not sufficient in traditional elections: this trust is usually only limited to the voting stage but not the tallying, and electors have no means to check if their vote was included in the tally, the tally cannot be recomputed by individual electors, etc. Additionally, traditional elections are tremendously expensive to organize, as they require in-person interactions in thousands of physical places distributed across the territory, and they do not handle properly the possibility of remote voting. In these remote scenarios, these security and audibility guarantees get degraded as there is a centralized and opaque process that handles those votes.

Electronic voting is proposed as an alternative to traditional voting. These systems generally make use of cryptography to formally distribute the responsibility between as many parties as possible while ensuring the integrity of the election process. The use of cryptographic commitments ensures parties act honestly, and the use of encryption preserves the users' privacy. These new mathematical guarantees open the door to new possibilities: universal verifiability, privacy, and remote and multi-device voting.

So far we described national elections, because it is the most pictured scenario when we discuss elections and voting, but the low cost of electronic voting enables the same systems to also accommodate all sizes of elections: regional, city council, or even private elections for companies or shareholders. Nonetheless, electronic voting also brings new problems: security, which is largely covered in Section 4, and usability.

We already covered that a huge factor in collaboration is trust. And while traditional elections have many flaws, people are accustomed to them and understand the general procedures. However, many people are not that familiar with cryptography and discrete mathematics, and this creates an entry barrier to electronic voting for most electors. They find electronic voting systems complex to use, and this creates a lack of trust that has affected electronic voting schemes in many instances (e.g. Switzerland Gerlach and Gasser [2009] or Estonia Maaten [2004], Drechsler and Madise [2004]), despite providing better security and audibility assurances. For this reason, in this dissertation, we focus on bringing traditional parties into the electronic process. So that electors can see these protocols, as the natural evolution of traditional systems with better securities, and not as a fuzzy competing approach.

1.3 Anonymous Identification & Access

Identification is another critical aspect of modern democratic societies. Identification refers to the process of identifying individuals, or entities, to an authority in charge of controlling the access to a service or the dealership of a product. There are many scenarios where a limited resource (e.g. ballots for voting, tax deductions), or a sensitive service (e.g. medical aid, financial or criminal records) need to be guarded. However, the identification, and later access to this guarded source, can benefit from being private. Here we describe some issues that can be mitigated by providing anonymous identification.

- Security: In some cases, revealing personal information can increase the risk of identity theft or fraud. Anonymous identification provides an additional layer of security by allowing individuals to access services without revealing sensitive information.

- Freedom of speech: In some countries, individuals may be at risk of persecution for expressing certain opinions or beliefs (e.g. political activism). Anonymous identification can allow individuals to express their opinions without fear of retribution. There are also many situations where users may feel more comfortable sharing their thoughts and opinions without revealing their identities.

- Discrimination: In some cases, individuals may be discriminated against based on their race, gender, sexual orientation, or other personal characteristics. Anonymous identification can help protect against discrimination by allowing individuals to access services or participate in activities without revealing personal information that could be used to discriminate against them.

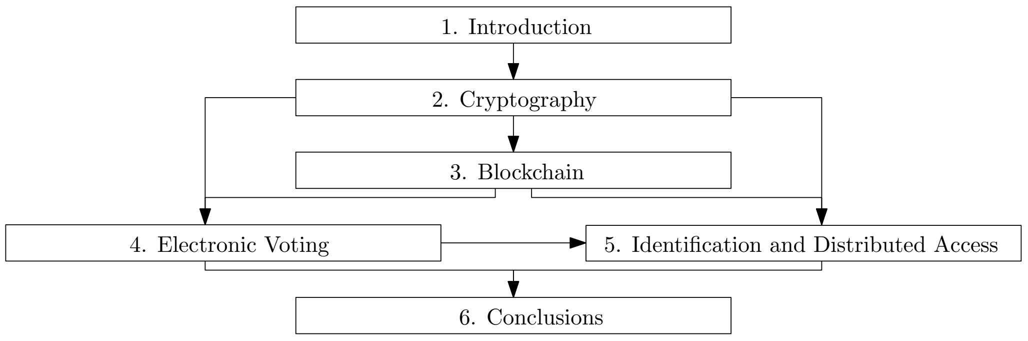
Individuals need to prove their identity in various situations, however, identity verification is not a simple process, and it involves many challenges. One of the most significant challenges is ensuring that individuals can prove their identity without revealing their personal information. Cryptography provides a solution to this problem by allowing the creation of anonymous identification systems that protect user privacy while ensuring only properly identified users get access.

In Chapter 5 we define the stages for the identification process (identification, registration, and access), and present three decentralized cryptographic protocols for anonymous registration and access to a service. We show how the anonymous registration process enables users to get access-keys that can be later used anonymously without being possible to relate in any way the access-key to the user. Therefore, users can anonymously access services and resources without compromising their privacy.

1.4 Thesis Organization

This dissertation is organized into six chapters, related as depicted in the next Figure. And the content of each chapters it is summarized here:

Chapter 1 summarizes the goal of this thesis, and introduces the fields of electronic voting and anonymous identification and access. It reviews the main challenges of these areas and how distributed cryptography can be used to address them.



Chapter 2 describes the mathematical framework used to implement the cryptographic primitives used in our methods. It also addresses the cryptographic assumptions and computation models that will be relevant in later chapters.

Chapter 3 provides an introduction to distributed ledger technologies as a way to achieve decentralization. It presents the concept of smart contracts and privacy-oriented solutions for blockchain. It also compares the most relevant cryptocurrencies that will be later used in the thesis.

Chapter 4 describes the electronic voting approach as opposed to traditional voting, and describes the properties, every voting scheme should accomplish. We present here three different voting protocols that fulfill them under different security assumptions.

Chapter 5 addresses the open problem of anonymous identification. The problem of anonymous registration is presented, along with three different protocols that allow for anonymous access. All the protocols are evaluated and their security properties are proved.

Chapter 6 draws the fundamental conclusions of this thesis and enumerates the main academical results of our research.

Besides these chapters, the thesis also presents four complementary appendixes. Appendix A describes the implementation of one of the proposed electronic voting schemes described in Chapter 4 by leveraging blockchain technology introduced in Chapter 3.

Appendix B provides a Python-based implementation and an analysis of the ring signature scheme defined in Chapter 3.

Appendix C reviews the basic structures and methods to carry out an election in one of the methods proposed in Chapter 4.

Finally, Appendix D provides a complete implementation and empiric evaluation of the three identification protocols proposed in Chapter 5.

Chapter 2

Cryptography

Cuando hay clase, la velocidad es una minucia.

Carlos Ruíz Zafón

Ourvqo tap kyaee, ci iexotqqap ej caa yiecpim.

Vigenère Cipher, $k =$marina

Cryptography is the art of secure communication. The word cryptography is derived from two Greek words: *kryptós* (hidden) and *graphein* (writing). Together with Cryptoanalysis, which is the art of analyzing secret systems, they conform the Cryptology field that refers to the study of secret systems.

The need for secret conversations is as old as humanity itself. Many civilizations in many ages have tried to develop systems that allowed for a secure transmission of messages between authorized parties, while also protecting the messages from undesired attackers. From hiding messages in obscure places to physical rudimentary devices, to simple substitution ciphers, to codes, the story of humanity is swamped with cryptography use. The most comprehensive resource that covers the history of cryptography from ancient civilizations to the modern age is Codebreakers Kahn [1996].

Modern cryptography, which starts in the middle of the 20th century with the Enigma machine, is profoundly correlated with the information age, mathematics, and computer systems. Enigma machine is considered one of the first examples of computational security as opposed to traditional substitution ciphers. The latest era of cryptography, not only allows for secure communication between established parties. But also enabled the possibility of establishing trust between multiple entities without third parties or hard assumptions. This has contributed to the decentralization and distribution of computation and has helped to build better systems where the responsibility and power no longer need to relapse in a single individual.

In this chapter, we only cover the most relevant mathematical foundations and cryptographic concepts to our research. All the concepts here reviewed will be relevant in the forthcoming chapters.

2.1 Algebra

2.1.1 Groups and Fields

Groups and fields are algebraic structures that present unique properties suitable for cryptography. Within this mathematical framework, arbitrary data can be represented as algebraic elements (encryption). And more importantly, represented back as the original data (decryption). This is possible only because groups and fields present a unique, and bijective, inverse operation. They also allow for efficient transformations on encrypted data.

Additionally, elements in this algebraic framework benefit from hard problems for which no efficient solution has been found, such as the Discrete Logarithm Problem (DLP) (see Section 2.5.1). These kinds of problems enable trapdoor functions in which most of public key cryptography is based (see Sec. 2.3). Thus, unless some secret values are revealed, algebraic elements are transformed into new elements that do not reveal much information about the original data they concealed.

Therefore, because of their efficient representation, and the existence of problems considered to be unfeasible to solve, groups and fields define the base of modern cryptography.

Groups

Groups were first introduced by the French mathematician Évariste Galois Galois and Neumann [2011] as a way to represent permutations. Groups define many symmetric properties that are useful in other mathematical objects. Hence, the concept of general group is defined as a set of elements G, and a binary operation $\odot$, that present the following properties:

1. G is associative i.e., $(a \odot b) \odot c = a \odot (b \odot c) \quad \forall a, b, c \in G$.

2. There exists a unique element $e \in G$, called the identity element, such that $a \odot e = e \odot a = a$.

3. $\forall a \in G$ there exists an inverse element a^{-1} such that $a \odot a^{-1} = a^{-1} \odot a = e$.

The group is represented as $(G, \odot)$, or simply G if there is no ambiguity with regard to the operation. Any group $(G, \odot)$ that accomplishes the commutative property i.e., $a \odot b = b \odot a$, is called *commutative* group or **abelian** group.

So far we have described groups using a general binary operation $\odot$. Usually, the operation can be as intuitive as either the multiplication or the addition of elements. Thus, the derived groups are usually referred to as multiplicative $(G, \cdot)$, or additive $(G, +)$ groups.

For the rest of this document, and unless stated otherwise, we assume multiplicative groups. For all multiplicative groups, there exists an equation that relates the inverse of the product of two elements $a, b \in G$:

$$(a \cdot b)^{-1} = b^{-1} \cdot a^{-1}$$

Subgroups

Let G be a group and H a subset of G. We say H is a subgroup of G, if:

1. $e \in G$, that is H contains the identity element of G.

2. H is closed under the group operation. If $a, b \in H$, then $ab \in H$ too.

3. H is closed under the inverse operation. If $a \in H$, then $a^{-1} \in H$ too.

Please note that we can always compute a group from an element $g \in G$ by computing its powers. Indeed, this is the smallest subgroup that contains g (as any group that contains g must also contain its powers) and we will say the group $< g >$ was generated by $g \in G$.

By Lagrange's theorem 2.1.1, the order of a subgroup H, divides the order of the group G. This is denoted as:

$$ord(H)|ord(G)$$

Theorem 2.1.1 (Lagrange)**.** *For any finite group G, of order $ord(G)$, the order of every subgroup of G divides $ord(G)$.*

Cyclic groups

Let G be a group, g an element of G, e the identity element of G and $n \in \mathbb{Z}$. If $n > 0$, we can define:

$$g^n = \underbrace{g \cdot g \cdot \ldots \cdot g}_{n \text{ times}}$$

If $n = 0$, we can see that $n \cdot g = e$. Finally, if $n < 0$, the multiplication can be expressed as:

$$g^{-n} = \underbrace{g^{-1} \cdot g^{-1} \cdot \ldots \cdot g^{-1}}_{|n| \text{ times}}$$

The set of all powers of $g \in G$ referred to as the subgroup generated by g, and as $< g >$. The same applies to additive groups, and the subgroup generated by g is called multipliers.

We call any group G, that consists only of powers of a single element g, **cyclic**. g is called the generator of G, and the whole group can be expressed as $G =< g >$. The order of $g \in G$ is denoted as $ord(g) = i$ and stands for the smallest integer i such that $g^i = e$.

2.1.2 Fields

A field F is an algebraic structure that comprises two binary operations $(+, \cdot)$, and satisfies the following properties:

1. $(F, +)$ defines an abelian group with an identity element denoted as 0.
2. $(F - \{0\}, \cdot)$ defines an abelian group with an identity element denoted as 1.

3. The operation $\cdot$ is distributive over $+$, this is, $a \cdot (b + c) = a \cdot b + a \cdot c \ \ \forall a, b, c \in R$.

In cryptography, finite fields, this is fields with a finite number of elements, are also referred to as Galois Fields, or simply GF. If we consider the set of integers $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$, called integers modulo p. We can be sure $\mathbb{Z}_p$ is a field if, and only if, p is prime, and we denote it as F_p.

We say that K is an extension field of F_p, if K contains F_p, $F_p \subseteq K$. F_p is also called a subfield of K. We can see extension fields as a vector space over the original field F_p. This implies that an extended field K contains $k-$dimensional vectors of elements in F_p. This is usually represented as $K = F_{p^k}$. Therefore, fields are always defined by a prime number, or the power of a prime number. Elements in F_{p^k} can also be represented as $(k-1)$-degree polynomials whose coefficients are elements F_p.

This extended representation of fields is extremely efficient for computation as we do not need to operate with hauling how we would do in the arithmetic of natural numbers, and we can be sure the elements remain within the field. One of the best uses of these properties is the Advanced Encryption Standard (AES) Nechvatal et al. [2001], where computations are carried out in the finite field extension F_{2^8}, also referred to as $GF(2^8)$.

2.1.3 Polynomials

Polynomials defined over large finite fields are extremely useful for a multitude of cryptography tasks: secret sharing, errorcorrecting codes, data availability, and, zero-knowledge proofs, among some of them. Given its relevance, we provide a short formal definition for polynomials and cover some fundamental algorithms to operate with them.

Let F_p be a finite field defined by a prime p, any expression of the form:

$$f(x) = \sum_{i=0}^{d} a_i x^i \quad a_i \in F_p, \tag{2.1}$$

where d is an arbitrary positive integer, is called a polynomial over F_p. The set of all possible polynomials over F_p is denoted by $F_p[x]$.

All polynomials have a degree, denoted as $deg(f)$ which is computed as the highest power of x that appears within the polynomial.

Lagrange's Interpolation

In many scenarios, we need a method to recover a polynomial from a set of its points. One of the most popular methods is Lagrange's interpolation method.

Lagrange interpolation states that for a set of points

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_d, y_d)\}, \tag{2.2}$$

there exists a unique uni-variate polynomial $f(x)$ of degree at most $d-1$ such that:

$$y_i = f(x_i) \quad \forall i \in (1, 2, \dots, d) \tag{2.3}$$

To recover the polynomial $f(x)$ we can apply the following equation:

$$f(x) = \sum_{i=0}^{j} y_i l_i(x) \tag{2.4}$$

where $l_i(x)$, represents each one of the Lagrange basis polynomials.

$$l_i(x) = \prod_{\substack{0 \le k \le j \\ k \ne i}} \frac{x - x_k}{x_i - x_k} \tag{2.5}$$

2.1.4 Elliptic Curves

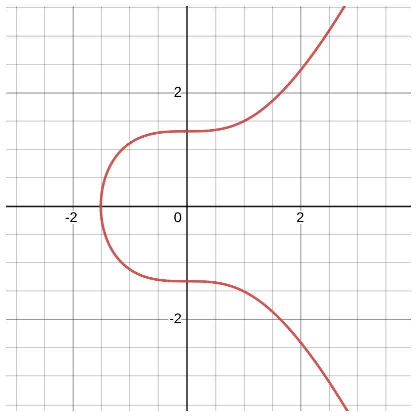
Diophantine equations Mordell [1969] are a branch of number theory that deals with the roots of polynomial equations in integer, or rational, numbers. Elliptic curves Koblitz [1987], Miller [1985] are a kind of cubic curves that, when defined over a finite field, fall under the definition of diophantine. Let F_p be a finite field defined by a prime p, and $a, b \in F_p$ scalars such that the cubic curve $X^3 + aX + b$ has no multiple roots. An elliptic curve over a field $E(F_p)$ is defined as the set of solutions $(x, y) \in F_{p^2}$ that satisfy the equation:

$$Y^2 = X^3 + aX + b, \tag{2.6}$$

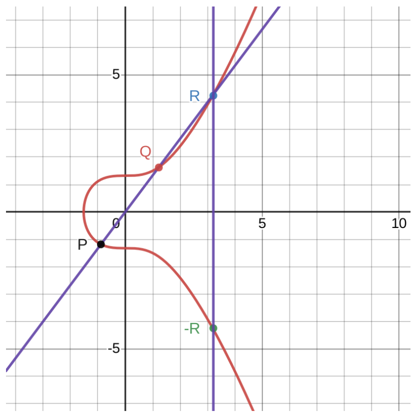
and a special point called "point at infinity" O. To be considered a valid elliptic curve, the cubic curve must be non-singular i.e.: all points must have a defined tangent. Hence, the discriminant d of the curve must be non-zero:

$$d = -(4a^3 + 27b^2) \ne 0 \tag{2.7}$$

Elliptic curves present two main geometric properties:



(a) Secp256k1 elliptic curve.



(b) Line intersecting elliptic curve.

Figure 2.1: Elliptic curve examples.

- An elliptic curve is symmetrical with respect to the x-axis. Thus, given a point $P = (x, y)$ on $E(F)$, we can be sure $-P = (x, -y)$ is also contained in the curve.
- Any straight line will intersect with an elliptic curve in at most 3 points (See Figure 2.1b).

Using these properties, the addition operation is defined so that the sum of three aligned points results in the neutral element. Thus, the addition of three aligned points, and the identification of the point at infinity with the neutral element, can be expressed as $P + Q + R = 0$. Therefore, the addition of the first two points is equal to the reflection of the third point with respect to the x-axis, $P + Q = -R$. See Figure 2.1b for a graphical representation. We refer the reader to Slinko [2020], Silverman and Tate [1992] for the demonstration.

An additive abelian group can be built using the points in the curve, and the point at infinity. This implies we can also find a generator G, as defined in Sec. 2.1.1, that generates the whole set of points in the curve. This group, and elliptic curves in general, are especially interesting for the field of cryptography since the DLP is also present (referred to as the Elliptic DLP or EDLP), and we can build on top of it to construct secure cryptographic systems. Elliptic curves also provide an equivalent security level compared to other structures, with a lower key size, because the cutting-edge results to solve the DLP in usual groups are not suitable to be applied in elliptic curve groups. This is especially relevant given the rise of general computing power available through the years.

2.2 Secret Sharing

Secret sharing schemes enable a secure method to distribute a secret between a set of parties, in such a way that the individual shares do not leak any information about the secret, and, when parties, or a subset of them, collaborate, they can recover the original secret. Adi Shamir Shamir [1979] introduced a (t, n)-threshold sharing scheme that allowed to share a secret C among n players, in such a way that any subset of $t + 1$ players can recover the secret C. The secret is encoded as the independent term of a polynomial $f(x)$ and the pieces of information distributed among players are points of the aforementioned polynomial.

$$f(x) = a_t x^t + a_{t-1} x^{t-1} + \ldots + a_1 x^1 + C \pmod q$$

Where t is the degree of the polynomial, C is the encoded secret, and q is a large prime.

To recover the secret C, any $t + 1$ parties need to interpolate the polynomial $f(x)$ from the points (x_i, y_i), we recommend Lagrange's interpolation (see Section 2.1.3). However, other methods based on the Discrete Fourier Transform (DFT) are also widely employed Gentleman and Sande [1966]. DFT introduces some assumptions on the periodicity of the function to interpolate and requires a change of basis, but these drawbacks are outweighed by its efficiency for large samples.

Please note that in Shamir's secret sharing scheme, the secrecy of C does not depend on the computational theory, but on information theory. Thus, the scheme has comparable security to Perfect Secrecy (see Sec. 2.5.3) systems, and it is resistant even in post-quantum scenarios.

2.2.1 Verifiable Secret Sharing

The scheme presented by Shamir relies on an honest dealer that generates partial shares and communicates them. The existence of such a dealer might not be guaranteed in many scenarios and leaves the parties helpless to a malicious dealer that sends invalid shares. Feldman Feldman [1987] introduced an extension to Shamir's work to address this matter. The scheme makes use of homomorphic properties in order to generate commitments of the polynomial $f(x)$ coefficients. Commitments take the form of:

$$c_0 = g^C, c_1 = g^{a_1}, \ldots, c_t = g^{a_t} \pmod q$$

Where g is a generator of the group G_p defined by prime p, and p is a second prime such that $q|p-1$. Any party can verify the correctness of its share with these commitments. Let (x_i, y_i) denote the share received by a concrete party. Parties can verify their share as follows:

$$g^{y_i} = c_0 c_1^{x_i} c_2^{x_i^2} \ldots c_t^{x_i^t} = \prod_{j=0}^{t} c_j^{x_i^j} = \prod_{j=0}^{t} g^{a_j^{x_i^j}} = g^{\sum_{j=0}^{t} a_j^{x_i^j}} = g^{f(x_i)} \pmod p$$

If the equation holds, the receiving party can be sure that the partial share is part of a valid secret-sharing scheme.

2.3 Public-Key Cryptography

Traditionally, cryptography was focused on symmetric encryption, hash functions, Message authentication codes (MAC), and other kind of secret-key protocols. Symmetric encryption imposes severe limitations in its applications and requires both parties to agree on a secret key. Due to the military nature of early cryptography standards and implementations, the use of secret-key cryptography was extensive and these limitations were not a major concern.

With the birth of the Internet in the 70s, the development of cheap digital hardware, and the declassification of military standards, the need for cryptographic systems that allowed to communicate unknown parties without trust assumptions, increased. In this context, and with the seminal papers of Diffie-Hellman and RSA, public-key cryptography was born. A new cryptographic framework, based on hard problems for which no efficient solution has been found (See Sec. 2.5.2). Public-key cryptography involves two families of algorithms, an encryption one, and a decryption one. These algorithms are related through a pair of keys s, v in the space of all possible keys $\mathcal{K}$, such that they represent invertible transformations over the finite space of possible messages $\mathcal{M}$. More formally:

- $\forall(s, v) \in \mathcal{K}$, s defines a transformation, and v defines the inverse of that transformation.

- $\forall (s, v) \in \mathcal{K}$, the encryption and decryption algorithms are easy to compute.
- For almost every pair $(s, v) \in \mathcal{K}$ it is computationally unfeasible to derive s from v.
- $\forall (s, v) \in \mathcal{K}$ it is feasible to compute inverse pairs v and s.

We cover the most relevant contributions for this thesis related to public-key cryptography.

2.3.1 Diffie-Hellman

In Diffie and Hellman [1976], Whitfield Diffie and Martin E. Hellman introduced a novel way to exchange a private key component over an insecure channel, removing this way the need for a previous key distribution process over a secure channel. And allowing the conversation between unrelated entities without trust assumptions or third parties.

Let F_q denote a finite field over a large prime q, and let g denote a generator on that field. Let A, B be two users wishing to establish communication with each other over a public channel.

1. Secret generation and commitment.
   (a) A generates a secret value $a \in F_q$ and computes g^a.
   (b) B generates a secret value $b \in F_q$ and computes g^b.
2. Exchange phase.
   (a) A sends $g^a \mod q$ to B over the channel.
   (b) B sends $g^b \mod q$ to A over the channel.
3. Obtaining the key.
   (a) A applies his own secret to the received message to obtain the key. $k = (g^b)^a = g^{ba} \mod q$.
   (b) B applies his own secret to the received message to obtain the key. $k = (g^a)^b = g^{ab} \mod q$.

The keys k obtained by A and B in steps 3(a) ad 3(b) are equivalent. Therefore, A and B established a unique private session key component for their communications.

Because of the DLP and the Decisional Diffie-Hellman assumption, neither of the parties, or an eavesdropper, can obtain the counter-party's secret (see Sec. 2.5.2).

2.3.2 RSA

RSA Rivest et al. [1978], named after their inventors Ronald Rivest, Adi Shamir and Leonard Adleman, is one of the most used public-key cryptographic systems. RSA is based on intractability of the integer factorization problem Montgomery [1994] and consists on three algorithms: key generation (Algorithm 1), encryption (Algorithm 2) and decryption (Algorithm 3).

Algorithm 1 RSA Key Generation: An entity generates a RSA private key component and its associated public key component.

```
1: (a) Generate two large and random primes $p$ and $q$.
2: (b) Compute $n = pq$ and $\phi = (p-1)(q-1)$.
3: (c) Choose a random integer $v, 1 < v < \phi$, such that $gcd(v, \phi) = 1$.
4: (d) Compute the unique integer $s, 1 < s < \phi$, such that $vs \equiv 1 \mod \phi$.
5: (e) The public key component is set to $(n, v)$, the private key component is $s$.
```

Algorithm 2 RSA Encryption: B encrypts a message m for A.

```
Require: $m \leftarrow$ Message to be encrypted.
Require: $(n, v) \leftarrow$ $A$'s public key component.
1: (a) Encode the message $m$ as an integer in the range $[0, n-1]$.
2: (b) Compute $c = m^v \mod n$.
3: (c) Send ciphertext $c$ to $A$.
```

Since the same three algorithms can be used to both encrypt and sign (depending on the component of the key used), RSA offers great flexibility. RSA also presents some security issues that need to be taken into account when implementing the scheme.

Algorithm 3 RSA Decryption: A decrypts a ciphertext c from B.

```
Require: $c \leftarrow$ Ciphertext for message $m$.
Require: $m \leftarrow$ Message that was encrypted.
Require: $s \leftarrow A$'s private key component.
1:  (a) Recover $m$ using $s : m = c^s \mod n$.
```

- RSA is sensible to integer factorization Kleinjung et al. [2010]. Carefully selected parameters Burns and Mitchell [1994] are required to avoid the most common attacks, such as the common modulus attack DeLaurentis [1984].

- RSA is deterministic and encryption/signing always produces the same output for a constant input. If the space of possible messages is small, or an attacker's computational power is not bounded, he could explore which message produced a given ciphertext.

- RSA is homomorphic and malleable. Therefore, a malicious attacker could intercept a ciphertext c and compute $c^{'} = c2^v$, causing the decryption to be $2m$ instead of m as originally intended.

For this reason, and to solve the last 2 issues, implementations of RSA include a masking process, also referred as padding, by which the messages are extended with a random string or any other method (e.g. hash of the message) to prevent homomorphic properties to be applied. This removes the determinism from RSA and ensures that malleable tampering cannot go undetected.

2.3.3 Commitment Schemes

Commitments are a kind of cryptographic construct that allow to commit to a secret value, without revealing any information about the secret value. The general scheme of any commitment scheme is based on two steps:

- *Hide*: Given a secret input x, the user produces a commitment C such that, it does not reveal any information of x, and the user cannot modify x without affecting C.

- *Reveal*: The creator reveals some information, also called the decommitment string, which allows opening the commitment C and reveal x.

With this information, the verifier can check the integrity and validity of the commitment.

There are multiple commitments schemes with different properties. One of the most famous commitments schemes are Pedersen commitments Pedersen [1991]. Nonetheless, a general commitment can be defined using a hash function as an oracle function and some source of randomness r.

$$C(x, r) = H(r||x), \tag{2.8}$$

where H is a public hash function and r acts as decommitment string. Once r is made public, everyone can open the commitment and verify its integrity. If x is meant to be secret, the reveal of the commitment implies to unveil both x and r.

2.4 Zero-Knowledge

Zero-knowledge proofs Thaler [2022] are a special kind of mathematical proofs. A proof is defined as any piece of information that convinces someone from the validity of a statement. A proof-system is any set of predefined steps to check if a proof is correct or not. Proof systems must ensure that no false statement can produce a valid proof (soundness), and that any true statement should have a corresponding valid proof (completeness). Typically, in these interactive schemes, two parties are involved: someone that wants to prove a statement and generates the proof: the *Prover*. And someone that is yet to be convinced of the validity of the statement, and verifies the proof: the *Verifier*.

Zero-knowledge proofs introduce an additional restriction: the verifier should learn nothing from the prover other than the validity of the statement. Zero-knowledge intuition might be easy to grasp Quisquater et al. [1989], but it is an extensive area of cryptography with thousands of applications. From interactive proofs Goldwasser et al. [2019], Babai [1985], Goldwasser et al. [2015], to specific domain zero-knowledge proofs, to the raise of general purpose zero-knowledge schemes.

Domain specific proofs provide zero-knowledge for some concrete scenarios: proving discrete logarithm solutions are known for a given instance (see Section 2.4.1), or that a secret value lies within a defined range Morais et al.

[2019]. General purpose zero-knowledge schemes use an intermediate arithmetic representation to represent arbitrary statements. Despite their heavy computational requirements, general purpose zero-knowledge schemes have growth extensively Parno et al. [2016], Chiesa et al. [2020], Maller et al. [2019], Groth [2016], Gabizon et al. [2019] and made it from theoretical research to real case scenarios, specially in the blockchain environment.

In this section, we summarize one of the most used proofs in e-voting and a transformation to make zero-knowledge proofs non-interactive.

2.4.1 Schnorr Zero Knowledge Identification

Schnorr introduced a simple, interactive and efficient zero knowledge scheme Schnorr [1989] to prove the solution to an instance of the DLP. Schnorr's proof is limited to the DLP problem, but it is extensively used in secure multi-party computation Canetti et al. [1996] or key-exchange scenarios where parties need to prove the validity of their partial values without revealing them.

Let us assume a prover needs to identify himself by proving he knows the discrete logarithm x of some group element $h = g^x \in G_q$. Where G_q is a group defined by a prime q. The group G_q and its generator g are public and known values. h is stored in the service side to properly acknowledge the identity of the user. Algorithm 4 depicts the steps of the identification protocol used to access a resource.

Schnorr's protocol is also sometimes referred as the sigma protocol, because of the pattern of the interaction between the verifier and the prover. As shown in Figure 2.2, the flow presents similarity to the sigma (Σ) Greek letter. But sigma protocols are a broader definition, usually referred to interactive proofs.

Nonetheless, the protocol presents a serious limitation: it is not a complete Zero-Knowledge proof. It is an Honest-Verifier Zero-Knowledge proof Goldreich et al. [1998], meaning that zero knowledge is only preserved if the verifier remains honest. A malicious verifier could extract the witness from the response received from the prover. For this reason, Schnorr's proof is always used together with the Fiat-Shamir heuristic, which makes the proof no longer interactive and prevents a dishonest verifier from acting.

Algorithm 4 Schnorr's Zero-Knowledge identification protocol.

```
Require: g ← Generator of $G_q$.
Require: x ← Solution to the DLP for an element $h$, using $g$ as a generator
    that identifies the user.
 1: Prover
 2:     (a) Sample a random value $r \in_R G_q$.
 3:     (b) Compute $u = g^r \mod q$.
 4:     (c) Send $u$ to the verifier.
 5: Verifier
 6:     (a) Sample a random challenge $c \in_R G_q$.
 7:     (b) Send $c$ to the prover.
 8: Prover
 9:     (d) Compute $z = r + xc \mod q$.
10:     (e) Send $z$ to the verifier.
11: Verifier
12:     (c) Accept if, and only if, $g^z = uh^c \mod q$.
```

2.4.2 Non-interactivity and the Fiat Shamir Heuristic

Fiat-Shamir Fiat and Shamir [1986] heuristic provides a way to produce non-interactive schemes. Non-interactivity implies fewer messages between parties and ensures the challenge selected by the verifier is random. Fiat-Shamir replaces the challenge from step (a) Verifier in Algorithm 4, with a one-way hash function $hash : \{0,1\}^* \rightarrow \{0,1\}^k$ of k-length.

As it can be seen in Algorithm 5, the number of interactions required to identify the user are reduced, and the previously discussed vulnerability of Schnorr's is removed.

Fiat-Shamir transformation is not limited to this example and can be, and it is, to many other zero-knowledge proof systems. When using this in a protocol implementation, extraordinary care needs to be taken when deciding which variables constitute the input of the hash function. Leaving any value that can be chosen by the prover outside the challenge computation would result in well known vulnerabilities Bernhard et al. [2012], Haines et al. [2020]. This implementation flaw is commonly referred as *weak* Fiat-Shamir.

$$\begin{tikzcd}
\text{Prover}(x, h = g^x) & & \text{Verifier}(h = g^x) \\
r \in_R Z_q & & \\
u = g^r \arrow[rr, "u"] & & {} \\
& & c \in_R Z_q \arrow[dll, "c"'] \\
z = r + cx \arrow[drr, "z"] & & \\
{} & & g^z \overset{?}{=} uh^c \arrow[ll, "0/1"']
\end{tikzcd}$$

Figure 2.2: Schnorr's proof interaction scheme.

2.5 Complexity & Computability

Cryptography is deeply tied to theoretical computer science. There are two major theories in the theory of computation Sipser [1996]: complexity theory Hartmanis and Hopcroft [1971] and computability theory Cooper [2017]. Complexity theory studies how much resources, in terms of time and memory, it takes to solve a particular problem. Computability theory deals with what can and cannot be solved by a computer. Thus, it is possible to distinguish among those problems for which there exists an algorithm to solve them, known as decidable problems, and those that have no such an algorithm, which are referred to as undecidable.

Most modern cryptographic systems are based on the complexity theory. These systems employ mathematical problems with an elevated complexity, for which no efficient solution is known, to build trapdoor functions. A trapdoor function is a function that is easy to compute in one direction, but it is unfeasible to do in the opposite direction (the inverse) unless a secret value is known. However, the security of systems based on the complexity theory relies on the current state of available computation, i.e. accessible hardware,

Algorithm 5 Non-interactive Schnorr's Zero-Knowledge identification protocol.

```
Require: g ← Generator of $G_q$.
Require: x ← Solution to the DLP for an element $h$, using $g$ as a generator
    that identifies the user.
1: Prover
2:     (a) Sample a random value $r \in_R G_q$.
3:     (b) Compute $u = g^r \mod q$.
4:     (c) Compute $c = hash(g||q||h||u)$.
5:     (d) Compute $z = r + xc \mod q$.
6:     (e) Send $u, c, z$ to the verifier.
7: Verifier
8:     (a) Check if $c = hash(g||q||h||u)$.
9:     (b) Accept if, and only if, $g^z = uh^c \mod q$.
```

and on some unconfirmed hypothesis such as $\mathcal{P} \neq \mathcal{NP}$. Different kinds of cryptographic systems, based on the computability theory are possible Sempere [2002, 2004]. Hence, systems are based on undecidable problems.

In this section, we cover some of the most well-known problems and assumptions that make public-key cryptography possible. As well as we introduce the concept of perfect secrecy, an approach not based on complexity theory.

2.5.1 Discrete Logarithm Problem

The intractability of the Discrete Logarithm Problem (DLP) McCurley [1990] is the base for many cryptographic protocols.

Let G_p be a finite cyclic group of order p. Let g denote a generator of G_p, and let a be an element in the same group. The DLP of a to the base g is the unique integer x, $0 \leq x \leq p-1$, such that:

$$a = g^x \Longleftrightarrow x = \log_g a \mod p \tag{2.9}$$

When defined over elements of a group, there is no efficient solution for the DLP. Thus, many cryptographic schemes employ the DLP to build trapdoor functions that are intractable to solve under the complexity theory.

2.5.2 Cryptographic Assumptions

Most public key cryptographic schemes are based on problems, also called trapdoor functions, that are assumed to be difficult to solve without the knowledge of a secret value, These assumptions state that it is difficult to solve, or even differentiate, instances of the problem without the complete knowledge of the parameters. We present some of the most well-known and relevant assumptions in public key cryptography.

DDH: Decisional Diffie-Hellman

Decisional Diffie-Hellman is a computational hardness assumption about the discrete logarithm problem in cyclic groups. The assumption states the following:

> Let G_q be a cyclic group of prime order q, with a generator g. The following two distributions over G_{q^3}: $A = \{(g^a, g^b, g^{ab})$ for $a, b \in_R G_q\}$ and $B = \{(g^a, g^b, g^c)$ for $a, b, c \in_R G_q\}$ are computationally indistinguishable.

Computationally indistinguishable means that the value g^{ab} looks like a random element in G_q, even if we know g^a, g^b. DDH is strongly related to another assumption called Computational Diffie-Hellman, which states that given g, g^a, g^b is computationally intractable to compute g^{ab}.

These assumptions are stronger than the DLP, as there exist groups for which the DLP is considered to be hard, but the DDH problem is easy.

RSA & Strong RSA

RSA assumptions are computational hardness assumptions over the RSA problem. Regular, or weak, RSA assumption differs from the regular RSA assumption in that it allows the attacker to freely choose the exponent message m. Both assumptions imply that RSA works as a one-way trapdoor function and, unless the secret component is known, there is no efficient solution for the RSA problem:

> Given a RSA public key component (n, e) and a ciphertext $c = m^e \mod n$, recover the message m.

RSA Assumption

The RSA assumption states:

> It is hard to solve the *RSA* problem unless the secret component of the key is known when the modulus n is large enough and randomly generated for a random message m.

Strong RSA Assumption

The RSA strong assumption states:

> It is hard to solve the *RSA* problem unless the secret component of the key is known when the modulus n is large enough and for any possible message m.

2.5.3 Perfect Secrecy

Perfect secrecy is directly derived from the information theory Shannon [1949] proposed by Claude Shannon. Information theory relates the space of possible messages $\mathcal{M}$ with the space of possible ciphertexts $\mathcal{C}$. Each possible transformation between these spaces corresponds to enciphering with a particular key. Each transformation (i.e. key) and each message have an a priori probability of being chosen. The combination of these 2 probabilities represents the a priori knowledge of the situation for a given attacker. If the attacker intercepts the ciphertext, he could compute its a posteriori probability by exploring the set of possible messages and keys. This a posteriori probability represents the problem complexity of cryptanalysis.

If for a given message, its cryptogram reveals nothing about the original message (e.g.: its length), we obtain perfect secrecy. Or, as more formally defined in Equation 2.10, perfect secrecy implies that the a priori probability of a given message m, in the space of all possible messages $\mathcal{M}$, is equal to the a posteriori probability of the message given the ciphertext c, in the space of all possible ciphertexts $\mathcal{C}$.

$$P(\mathcal{M} = m) \equiv P(\mathcal{M} = m|\mathcal{C} = c) \tag{2.10}$$

and, thus, that the ciphertexts reveal no information about the message. All messages are equiprobable for a given ciphertext, making the scheme secure since the attacker has no method to obtain additional information, even with selected ciphertexts. Multiple solutions for a given ciphertext can exist. Hence, even an attacker with unlimited computing resources, could not guarantee the validity of a solution.

To attain perfect secrecy in a system, the secret key utilized must be equal to or longer than the message being encrypted Dodis [2012]. In this dissertation, we make use of polynomials defined over finite fields to achieve perfect secrecy. Within this framework, the length of the secret key can always be consistently expanded by modifying the prime p that defines the field. Therefore increasing the space $\mathcal{C}$ and finding longer secret keys to encode our message.

2.6 Digital Signatures

Digital signatures are a public-key cryptography primitive designed to provide guarantees of the authenticity of a message. They are the digital counterparties to traditional handwritten signatures. Digital signatures schemes usually consist in three different algorithms:

- Key generation$(1^\lambda) \to (s, v)$. As the algorithm that takes as input a security parameter λ and produces the signing key s and the public verification key v.
- Sign$(m, s) \to \sigma$. Which is the algorithm that takes a message m and the signing key s to produce a signature σ.
- Verify$(v, \sigma, m) \to 1/0$ As the verification algorithm that takes the message m and the signature σ and verifies them using the public key v. It outputs a boolean with the result of the verification.

Moreover, digital signatures offer certain attributes that surpass the capabilities of conventional signatures:

- Authenticity: The message originates from a legitimate party.
- Integrity: The message has not been tampered or modified.
- Non-repudiation: The signing party cannot subsequently deny the fact they actually signed the message.

The properties of public verifiability and non-repudiation, while often advantageous, may not be universally desirable. Numerous scenarios exist in which users might prefer to maintain their privacy (e.g., electronic voting) or restrict third-party verification of their signatures.

To address these concerns, innovative signing schemes have been proposed, frequently referred to as Non-Standard Arroyo et al. [2015] due to their unique architecture, that diverges from the traditional signer and verifier roles or the deviation from previously outlined properties. These novel, non-standard schemes include, but are not limited to:

- Blind Signatures. See Section 2.6.1.
- Group and Ring Signatures. See Section 2.6.2.
- Multi-signatures Itakura [1983], wherein multiple signers must sign the same message for it to be considered legitimate.
- Threshold signatures Desmedt and Frankel [1991], Damgård and Koprowski [2001], which necessitate a minimum number of parties to generate a single signature.

In the following sections, we cover in detail two of the most employed signatures schemes in this dissertation: blind signatures and ring signatures.

2.6.1 Blind Signatures

Blind signatures Chaum [1982], Camenisch et al. [1994] were introduced as a scheme to enable untraceable payment systems. Conceptually, blind signatures allow a user to get some data signed without revealing the actual information getting signed. Blind signatures enable an entity to verify the identity of a user and grant a certification without requiring the entity to see the actual data getting certified. This implies that later on, the properly identified user can make use of his signed data without being tracked.

Blind signatures take advantage of the homomorphic properties of cryptography to enable the signature of masked data. They can be defined over multiple cryptographic schemes, as long as they present these homomorphic features and provide the following functions:

1. A signing function $s(x)$ only known to the signer, and its corresponding inverse $s^{-1}(x)$ such that $s^{-1}(s(x)) = x$. Both functions must not reveal anything about the original message x.

2. A commuting function $c(x)$, and its inverse $c^{-1}(x)$, only known to the user such that $c^{-1}(s^{-1}(c(x))) = s^{-1}(x)$. Both functions must not reveal anything about the original message x.

3. A redundancy predicate r, ensures sufficient redundancy to make the search space for valid signatures impractical.

The main property of blind signatures is the inability to relate the signature $s(x)$, with the correspondent masked signature $s(c(x))$. For this reason, blind signatures are especially relevant in electronic voting schemes. Consult Section 4.2 to see an RSA implementation of blind signatures applied to voting.

2.6.2 Ring Signatures

Ring signatures are a distinctive form of cryptographic signature, named after their unique ring structure. Ring signatures utilize the public keys of multiple individuals to generate a signer-ambiguous signature, where the identity of the signer is not revealed, only their membership to a specific group. As any public key within a ring signature could potentially be the signer, the signature does not uniquely identify the originator.

Ring signatures were initially proposed by Chaum in Chaum and van Heyst [1991] as a form of group signature. However, this scheme was limited as it required a group coordinator to set up the signing scheme. To overcome this limitation, Rivest introduced the first ring signatures in Rivest et al. [2006], which provided unconditional signer ambiguity without the need for a group coordinator. In a ring signature scheme, any user can define a set of possible signers, including themselves, and sign a message using their own secret key and the public keys of other users in the set, without requiring the permission or knowledge of those other users. This provides unconditional signer ambiguity without the need for a group coordinator. However, since it is impossible to determine which of the users in the set actually signed the message, nothing would prevent a malicious user from using a different ring signature to double spend a resource, as the signature would still be considered valid as long as it was signed by a member of the defined set. For this reason, ring signature schemes that introduced some mechanism to prevent double signatures were introduced Wang et al. [2008], Chow et al. [2005]. See Section 3.3.2 for an implementation of Ring Signature Confidential Transactions, which leverage these mechanisms.

Chapter 3

Blockchain

Mi infancia son recuerdos de un patio de Sevilla.

Antonio Machado

Mq ukpbccda iou xteooeffc bk hb jkbwu lq Dcftblv.RF

Hill Cipher mod 27, Matrix $\begin{bmatrix} 2 & 2 & 3 \\ 4 & 3 & 6 \\ 5 & 8 & 13 \end{bmatrix}$

3.1 Blockchain Basics & Bitcoin

A blockchain is a data structure composed of blocks, that are stored sequentially and, therefore, define a chain. These blocks contain ordered lists of transactions that reflect transactional information between the parties involved in the blockchain. Processing these transactions sequentially allows us to grasp the current state of the blockchain. The state of the blockchain is defined as the number of tokens in each one of the addresses at a given time. For this reason, blockchains are sometimes also referred to as public, and decentralized, ledgers. Blocks are atomic in the sense that, either they are processed and a new state is achieved, or they are not processed and the blockchain state remains the same. Addresses act like accounts and are managed through keys. The owner of the private key associated with the address, has complete ownership of the funds and resources in the derived addresses.

The story of blockchain is slightly tied with Bitcoin Nakamoto [2008] and cryptocurrencies in general. The proposal of a peer-to-peer electronic cash network by Satoshi Nakamoto was the first to use blockchain technology in a decentralized environment and to solve the problem of consensus. In Bitcoin, parties collaborate to maintain the state of the payment network by using the blockchain as a distributed ledger that contains all the public information. This transactional information is used to compute the state and address of who owns what at a given block height. By giving users the ability to actively participate as first-class citizens in the network, and managing their own funds through pseudonymous addresses, Bitcoin was able to introduce a distributed payment network without any trusted party involved.

Nonetheless, the biggest contribution made in the Bitcoin whitepaper was solving the consensus problem. In an adversarial network with economic incentives where everyone has the same power, the obvious problem is how to prevent malicious parties from altering the state. Bitcoin built on top of previous electronic cash systems Wei Dai [1998], Szabo [2005] and reduction spam mechanisms Finney [2004] to develop Proof-of-Work. Proof-of-work (PoW) is a consensus algorithm that requires some computations to be done before submitting a block to the blockchain. This creates a trial-and-error race between parties to be able to add a block. This computational process is called mining. Assuming that at least 51% of the network is honest, the blockchain will be filled with valid blocks. In the case of a bifurcation, e.g. two parties mine a block at the same height, users should always follow the longest chain, the chain with more work. Proof-of-Work chains become more secure as the blockchain grows since it would require a malicious attacker to rewrite many past blocks to change the current state. Other famous consensus algorithms are Proof-of-Stake (PoS) Nguyen et al. [2019], where the block finalization depends on a validator set that is chosen based on their stake, and Practical Byzantine Fault Tolerance Bano et al. [2019].

3.1.1 Blockchain Trilemma

All distributed systems, no matter if it is a blockchain or a different system, fall under the CAP theorem Gilbert and Lynch [2002], Brewer [2012]. CAP theorem states that any distributed system, cannot provide a strong consensus (C), high service availability (A), and partition tolerance (P) at the same time. Since availability and partition tolerance are binary properties, they are ensured or not, no middle-ground is possible. Usually, consistency is the degraded property. This degradation results in different consistency models Muñoz-Escoí et al. [2019]. We already covered how Bitcoin, and other blockchains, resolve the consensus problem by using PoW. Nonetheless, PoW is not a strong consensus algorithm, which affects the finality of the transactions (see Section 3.1.2) and the security and efficiency of the network.

A problem derived from the CAP theorem, that affects all blockchains, no matter the consensus algorithm employed, relates to their scalability, security, and decentralization properties.

Security as the property that all transactions are correct and validated, and the blockchain cannot be manipulated.

Scalability defines the ability of the system to support more users without degrading the quality of the service.

Decentralization ensures that enough diverse entities participate in the network and redistribute the power among users.

The problem is known as the blockchain trilemma and states that in the triangle defined by the mentioned properties (see Figure 3.1), only two of them can be provided with sufficient guarantees, and with the cost of degrading the third property. Hence, every design decision in one direction is a trade-off in the spectrum of these properties. This means, that we can only focus on one side of the triangle.

Therefore, we can classify blockchains into three different categories:

- Those chains that focus on a strong consensus algorithm and provide great security, while also ensuring the decentralization of the system. The main drawback of these systems is their poor scalability, as more users engage in the system, the costs and times increase significantly. An example of this category is the Bitcoin network.

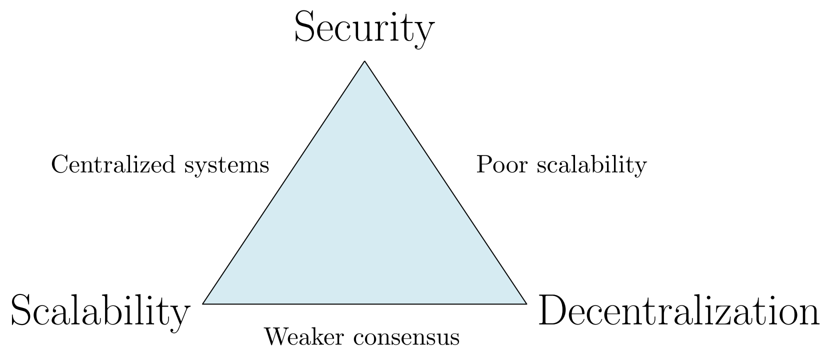


Figure 3.1: Blockchain Trilemma Problem

- Blockchains where the security is ensured and have great scalability solutions, but sacrifice the decentralization of the systems. They usually have a very reduced validator set that makes them pretty centralized. An example of this kind of network is Binance BSC.

- Systems where scalability is better, and they are still considered decentralized, but the security of the consensus algorithm is usually dependent on economic guarantees instead of the statistical unfeasibility of attacking the network. A possible example of this is the Ethereum network, as it employs PoS instead of PoW.

3.1.2 Block Finality

Depending on the consensus algorithm employed, blockchains provide different types of finality. By finality, we refer to the amount of time, or blocks in the chain, needed to consider the submitted transactions as immutable, or at least statistically unfeasible to modify. Finality simply represents the process of inclusion in the blockchain and guarantees that past events on the blockchain are immutable. Unfortunately, strong finality cannot be provided without some compromises, due to the CAP theorem, and most blockchains only offer some degree of finality. We list the three degrees of finality most likely to be found on different networks, from weaker to stronger finality:

- **Probabilistic finality:** Finality is reached eventually. Under some assumptions, we can estimate the probability that a given block is considered final. With each new block added to the chain, older blocks become more final. E.g. Bitcoin, and most PoW chains, consider a block final after 6 blocks since the probability of a fork decreases exponentially as the chain grows.

- **Provable finality:** In an effort to provide stronger and faster finality, some chains include some kind of finality gadget that runs in parallel to the chain, and performs a Byzantine agreement process over the blocks already in the chain. Once the gadget has gone over those blocks, and a consensus is reached, they are considered final. Some examples are GRANDPA[1] on Polkadot, or Casper FFG [2] on Ethereum.

- **Absolute finality:** At a cost, some blockchains implement Probabilistic Byzantine Fault Tolerant (PBFT) consensus protocols. This means, once the block is crafted, it is automatically considered final.

3.1.3 Addresses

The state of a blockchain is determined by transactions, and this state is represented using addresses. Addresses are controlled through digital keys and represent different pseudonymous identities. Addresses can own both cryptocurrencies and code in some blockchains.

Different blockchain networks decide to handle addresses differently. There are mainly two general approaches when handling addresses abstraction:

- **Unspent Transaction Output (UTXO)**: Transactions are represented as the sum of their inputs and outputs. Every transaction must prove that the sum of its inputs is equal to or larger than the sum of its outputs. Each input must be tagged as not yet spent, and the signature of the transaction must match the owner of the inputs. Unspent amounts in the transaction are returned to the owner as special output called unspent transaction output which gives a name to this approach. This is the model used in Bitcoin and Monero.

- **Account model**: In this model, addresses are modeled similarly to bank accounts. Each one of them has its own storage and balance. Any block in the chain can alter the state of one of the accounts. A

[1]https://github.com/w3f/consensus/blob/master/pdf/grandpa.pdf
[2]https://arxiv.org/pdf/1710.09437.pdf

transaction is valid if the account has enough balance for the transfer. This accounting model is used in Ethereum.

The account model is easier to manage and allows for more complex interactions. However, since any block can affect the account, you need to synchronize the whole blockchain in order to verify a transaction. UTXO model allows to verify a signature given only the inputs and outputs, it provides slightly better security guarantees as it becomes difficult to link multiple addresses, and allows for parallel transaction processing. Nonetheless, it is more limited and imposes severe restrictions in more complicated scenarios such as the use of smart contracts.

3.2 Ethereum

We devote this section to provide an introduction to the Ethereum ecosystem. An interested reader can find a comprehensive introduction to Ethereum in Antonopoulos and Wood [2018].

Ethereum was developed by Vitálik Buterin originally as a Bitcoin improvement. But the changes introduced were too ambitious and ended up being a different blockchain. The main changes introduced by Ethereum with regard to Bitcoin were:

- The transition from a PoW consensus to a PoS. Now blocks are added by a set of validators in each epoch of the network. Validators need to stake a sufficient amount of tokens that provide economic guarantees to make sure they validate blocks correctly. Please note that originally Ethereum was also using PoW, but later transitioned to PoS.
- The use of the account model for abstracting addresses as defined in Section 3.1.3.
- The introduction of smart contracts. Ethereum allows uploading arbitrary code, written in the language they created: Solidity, in the blockchain. This enables public, decentralized, and auditable code. The blockchain is no longer a simple transactional network, but a computing platform in which we can develop much more complex protocols.

Ethereum can be conceptualized as a global and decentralized state machine, wherein the state is established by a sequence of confirmed blocks, and new block proposals dictate modifications to the state. Each block consists of a set of transactions that define it. Consequently, Ethereum functions as a distributed virtual machine. The Ethereum yellowpaper Ethereum [2022] outlines the technical specifications of the Ethereum Virtual Machine (EVM), which serves as the mechanism responsible for processing transactions and updating the current state. To function effectively within the Ethereum ecosystem, all applications and smart contracts must comply with EVM standards. The EVM constitutes the core technical specification that manages the blockchain state, while Ethereum typically encompasses a broader scope, such as client node software, external data structures, and other associated components.

The Ethereum ecosystem features two distinct types of addresses: Externally Owned Accounts (EOAs) and contract addresses. EOAs are owned and managed by users who can control them using digital private keys, signatures, and address derivation techniques. In contrast, contract addresses are exclusively managed by the smart contract code itself. Consequently, contract addresses do not possess any associated private keys, and all operations on them are initiated by transactions originating from EOAs. However, an EOA address can own a contract address and make changes to the code. If a smart contract developer wants to provide a security guarantee of the immutability of the code, he can transfer the ownership to a special address (all zeroes address). This process is known as renouncing ownership of a contract and ensures not even the original developer can apply changes to the code.

3.2.1 Gas Fees

Ethereum employs gas as the unit of measurement for computational and storage resources. This system provides economic incentives for validators to operate nodes on the network and prevents the abuse of the blockchain through techniques such as infinite loops. As a result, users are required to pay a fee for each interaction they have with the network. Consequently, optimizing gas usage has become a crucial area of research that heavily influences smart contract development. To mitigate the prohibitive expense associated with certain operations, Ethereum includes a reduced set of pre-compiled contracts specifically designed for commonly used and computationally expensive mathematical operations, such as elliptic curve arithmetic, hashes, and signature verification. Without these pre-compiled contracts, implementing these operations would be excessively costly.

Gas fees are paid in Ethereum's native currency, ether (ETH), and are denoted in gweis. A gwei is equivalent to 10^{-9} ETH. Traditionally gas fees were calculated by multiplying required gas units and their gas price, $\text{Fee} = \text{Gas Units} \cdot \text{Gas price}$. As an example, assuming 21,000 units of gas, at a gas price of 100 gwei and an Ethereum price of 1400\$:

$$\text{Fee} = 21,000 \cdot 100 = 2,100,000 \text{ gwei} = 0,0021 \text{ ETH} \approx 2.96\$$$

After the London update of the network Roughgarden [2020], the gas price is divided into a Base fee, that gets burned to make the currency deflationary, and a Priority fee that is sent to miners.

3.2.2 Events

It is important to differentiate between accessibility and public availability. Ethereum leverages events and indexed parameters to optimize the querying of the blockchain and enable seamless integration with user interfaces. Without such optimizations, blocks, and transactions would be represented solely as hexadecimal strings that would need to be parsed manually.

Events are used to enable asynchronous triggers and provide context-specific data related to the event. User interfaces can monitor these events, which often include return values from transactions initiated by EOAs. Additionally, events offer a more cost-effective form of storage, in terms of gas usage, compared to standard smart contract storage mechanisms.

Events support up to three indexed parameters, allowing for precise indexing of events based on specific values. This feature facilitates the filtering of events with similar parameters from others with different values. For an in-depth illustration of the utility of events, please refer to our Solidity implementation of a voting protocol in Appendix A.

3.3 Monero

One of the key points in blockchain is the publicity of the information, anyone can verify the information and state of the blockchain. While great for transparency, this means that all the monetary transactional information can be consulted. If a user's identity is related, by any means, to his blockchain addresses, all his personal transactions will be traceable.

In an effort to provide better privacy solutions, while also ensuring that all data in the blockchain is still valid and can be audited, Monero was introduced. Monero Koe et al., Van Saberhagen [2013] is a PoW blockchain, under the UTXO model with no support for smart contracts, that leverages different cryptographic primitives to achieve untraceable transactions over a public ledger. In order to accomplish this, Monero employs three privacy-enhancing mechanisms:

- Zero-knowledge proofs (See Section 2.4), specifically Bulletproofs, to ensure the amounts transferred are within a valid range.
- One Time Public Keys ($OTPK$s), as defined in Section 3.3.1, to allow one-time addresses that protect the identity of the receiver.
- Ring Confidential Transactions (RingCT), as defined in Section 3.3.2, to hide the identity of the sender of a transaction.

3.3.1 One Time Public Keys

To anonymize the receiver of a transaction, Monero employs one-time addresses. One Time Public Keys ($OTPK$s) allow the sender to use a new address for each transaction. These new addresses are generated using a Diffie-Hellman exchange (see Section 2.3.1). Sending transactions using these keys preserves the anonymity of the receiver, as they cannot be related to their identity. The receiver, and only the receiver, is able to recover the private key of the one-time address, and therefore recover the funds.

In Monero, each user has two public keys (A, B). For any sender to make a transaction, he needs to compute the $OTPK$ that will be used in the transaction using the two public keys of the receiver. To do so, the sender also selects a random number r and computes:

$$OTPK = H_s(rA)G + B \tag{3.1}$$

Where H_s is a hash function that maps to integers within the elliptic curve field used in Monero.

The derived $OTPK$ is used as the public address of the transaction. Note that $R = rG$, is also added to the transaction payload. Once the transaction is completed, the receiver will be able to recover the private key x associated with the $OTPK$ using his own private keys (a, b):

$$x = H_s(aR) + b, \tag{3.2}$$

such that:

$$OTPK = xG = (H_s(aR) + b)G \tag{3.3}$$

It can be seen that Equations 3.1 and 3.3 are equivalent.

$$\begin{aligned}
H_s(rA)G + B &= (H_s(aR) + b)G \\
(H_s(rA) + b)G &= (H_s(aR) + b)G \\
(H_s(raG) + b)G &= (H_s(arG) + b)G \\
(H_s(aR) + b)G &= (H_s(aR) + b)G
\end{aligned}$$

3.3.2 Ring Signature Confidential Transactions

To anonymize the sender of a transaction, Monero employs ring signatures to provide signer ambiguity. As covered in Section 2.6.2, some ring signature schemes present the problem of double signing. In Monero, that would imply the ability to double spending a token. are used to sign transactions with a group of keys, thus giving ambiguity to the sender. In order to prevent that, Ring Signature Confidential Transaction algorithm Noether [2015], Van Saberhagen [2013] was introduced. This signature scheme combines the modifications for reducing space consumption described by Back [2015] and the introduction of Key Images. Key images are a public commitment of the signer's private and public keys. They do not leak any information about the signer's private key but allow to anonymously link the private key of the signer. Independently of the public keys on the ring, to the signature. This allows to prevent the use of the same key to sign two different rings. Therefore eliminating the possibility of double spending.

Algorithms 6 and 7, formally detail the generation and verification of RingCT signatures. In the algorithms, H_s represents a hash function that maps to integers within the elliptic curve field, and H_p is a second hash function that maps to elliptic curve points. Please note that because the ring signature generation involves random coefficients, and also depends on the signer's private key, the generation algorithm is not deterministic. For an implementation of these algorithms, please see Appendix B.

Algorithm 6 Signature Generation

```
Require: $N \leftarrow$ Size of ring signature.
    $P = \{pubk_1, pubk_2, \dots, pubk_N\} \leftarrow$ List of public keys used in the signature.
    $s \leftarrow$ Index in $P$ where the public key of the signer is stored.
    $x \leftarrow$ Signer's private key paired with the public component $pubk_s$.
    $m \leftarrow$ Message to sign.
 1: Let $r$ be a list of random numbers empty in the $s$ index.
 2: Let $\alpha$ be a random number.
 3: Let $L, R, c$ be empty lists.
 4: $K \leftarrow xH_p(pubk_s)$
 5: $L_s \leftarrow \alpha G$
 6: $R_s \leftarrow \alpha H_p(pubk_s)$
 7: $c_{(s+1)\ mod\ N} \leftarrow H_s(m, L_s, R_s)$
 8: $i \leftarrow (s+1)\ mod\ N$
 9: while $i \neq s$ do
10:     $L_i \leftarrow r_i G + c_i pubk_i$
11:     $R_i \leftarrow r_i H_p(pubk_i) + c_i K$
12:     $c_{(i+1)\ mod\ N} \leftarrow H_s(m, L_i, R_i)$
13:     $i \leftarrow (i+1)\ mod\ N$
14: end while
15: $r_s \leftarrow \alpha - c_s x$
16: return $(P, K, c_0, r)$
```

3.4 Other Blockchains and Applications

Blockchain ecosystems have flourished rapidly in the last decade. They have transitioned from niche and simple transactional networks, to mature and established decentralized systems with many applications. In this section, we provide a brief high-level overview of the most well-known networks and their most popular applications.

3.4.1 Other blockchains

We can roughly classify blockchains by the goal they were created to solve.

First generation blockchains Their goal is to develop a distributed payment network that does not depend on third parties. A decentralized

Algorithm 7 Signature Verification

```
Require: $N \leftarrow$ Size of ring signature.
    $P = \{pubk_1, pubk_2, \dots, pubk_N\} \leftarrow$ List of public keys used in the signature.
    $K \leftarrow$ Key image associated to the signature.
    $c_0 \leftarrow$ Signature seed that bootstraps the validation algorithm.
    $\{r_1, r_2, \dots, r_N\} \leftarrow$ List of random coefficients employed in the generation algorithm.
    $ListK \leftarrow$ List of already employed key images.
    $m \leftarrow$ Signed message.
 1: if $K$ in $ListK$ then
 2:     return $False$
 3: end if
 4: Let $L', R', c'$ be empty lists.
 5: $L'_0 \leftarrow r_0 G + c_0 pubk_0$
 6: $R'_0 \leftarrow r_0 H_p(pubk_0) + c_0 K$
 7: $c'_1 \leftarrow H_s(m, L'_0, R'_0)$
 8: while $i < N$ do
 9:     $L'_i \leftarrow r_i G + c'_i pubk_i$
10:     $R'_i \leftarrow r_i H_p(pubk_i) + c'_i K$
11:     $c'_{(i+1)\ mod\ N} \leftarrow H_s(m, L'_i, R'_i)$
12:     $i \leftarrow (i + 1)$
13: end while
14: return $c'_0 == c_0$
```

cash system that everyone can employ. Bitcoin and Monero are examples of this first generation blockchains.

Second generation blockchains aim to develop a smart contract layer on top of the distributed network. This enables third parties to develop applications and services, leveraging the chain as a settlement layer. Good examples of this category are Ethereum or Solana[3] networks.

Third generation blockchains also support smart contracts but are focused on network interoperability. This is connecting different blockchain networks efficiently without introducing new security assumptions. Some

[3]https://solana.com/es

examples of these kinds of networks are Polkadot[4], which runs multiple parallel chains connected to a central relay chain, and Cosmos[5] that connects chains through their own trustless communication protocol called IBC[6].

3.4.2 Applications

Smart contracts opened a new and Turing-complete space for application development. Many projects have benefited from this opportunity and have brought traditional applications to the blockchain environment, and created novel ones:

- Decentralized Finances (DeFi): This involves any traditional finance service that is not implemented as a smart contract: investing incentives, insurance, exchanges, lending, etc. Notable DeFi projects include: Lido[7], Curve[8], and Maker[9].

- Games: Entertainment has also made use of the blockchain as it ensures a public amount of resources, fair competition, and blockchain collectibles. Main contributors in this category are: Axies infinity[10], and CryptoKitties[11]

- Non-Fungible Tokens (NFTs). Fungibility refers to the ability to exchange a token for another one assuming they are equal, or at least similar. A non-fungible token is unique, and therefore cannot be considered equal to other tokens. NFTs are usually related to artistic expression. Some examples are: OpenSea[12], and CryptoPunks[13].

[4]https://polkadot.network/
[5]https://cosmos.network/
[6]https://ibcprotocol.org/
[7]https://lido.fi/
[8]https://curve.fi/
[9]https://makerdao.com/es/
[10]https://axieinfinity.com/
[11]https://www.cryptokitties.co/
[12]https://opensea.io/
[13]https://www.larvalabs.com/cryptopunks

3.5 Risks

In comparison to numerous well-established, time-tested, and extensively deployed technological advancements, the advent of blockchain technology remains relatively recent. Given its novelty, and the fact that is a decentralized network with many participants of varying honesty, it involves certain risks. While this factors do not impose unsolvable barriers, it is crucial they are taken into consideration when applying blockchain to some scenarios. We briefly enumerate the most notorious hazards for blockchain and other distributed ledger technologies.

- Consensus can be compromised, both Bitcoin and Ethereum have experienced reorganization attacks in the past Saad et al. [2020]. When using blockchain, you are inherently trusting its consensus mechanism and security assumptions. If that fails, it represents a supply-chain attack to your application.

- Being completely public might entitle challenges in many scenarios. Some cryptographic techniques, as the zero-knowledge proofs reviewed in Section 2.4, can be used to obtain privacy. However, these mechanisms do not prevent the user from willfully revealing their own identities.

- Key management is complicated and relies entirely on the good practices of users. Misplaced or lost keys may result in lost funds, inability to access services and sign digital messages.

- The existence of vulnerabilities in smart contracts is well-documented Destefanis et al. [2018], and these weaknesses can be exploited at a minimal cost for substantial gains. Such attacks may entail stolen funds, address impersonation and unauthorized access to user wallets.

- The infrastructure required to access blockchain can be compromised and used to trace user's identity or enable Man-in-the-Middle attacks.

- Modifications and updates to blockchain consensus mechanisms are often more time-consuming to implement than those in permissioned networks. These network updates usually necessitate an on-chain voting process, a hard fork and the majority of the network to accept them before they can be applied.

Chapter 4

Electronic Voting

A man can never have too much
red wine, too many books, or
too much ammunition.

Rudyard Kipling

2C363A4A42775BA3B191FDF3
D0D49AFA294BE4B09A969842
D889D06C3C58F4E14EA402253
9A235B4410D889175F91040A4E
28650C4EC27BB0AA6CDF340E
EE2EFCFBB69997BE6F625424C
4B944296C987

AES, k=RudyardKipling12
CBC mode

Voting is a crucial part of our societies. Elections, and the guarantees that they must provide, are the cornerstone of our democratic states. They affect all the aspects in our lives: social interactions, economic exchanges, organizational structures, personal finances, foreign relationships, etc. For this reason, all societies that have evolved into democratic structures have studied voting from different angles. Many different voting schemes exist, since multiple cultures and different situations require specific obligations.

We here present a short, non-exhaustive list of the most used and relevant voting systems Levin and Nalebuff [1995], Nurmi [2012]:

Plurality: The vote is the name of one candidate. Most voted candidate wins.

Approval Brams and Fishburn [2007]: You can approve as many candidates as you desire in a multi-candidate race. Most approved candidate wins.

Borda Saari [2012]: You rank the C candidates. Each one receives $C - K$ points for being ranked in the Kth position. Candidate with most points wins.

Condorcet systems Nicolas de Condorcet [1785]: Candidates are ranked. If a candidate exists that is preferred pairwise (Condorcet winner) over each other contender by the majority of the votes, then he wins. Since a Condorcet winner does not always exist, a number of techniques and heuristics exist to determine a winner given the CXC matrix defining the relations between candidates.

Instant runoff: Candidates are ranked, the one with the lower number of votes is eliminated from the election. This process iterates until just one candidate, the winner, remains.

Range voting: The vote is an integer within a fixed range. Each elector assigns a range for all the possible candidates. The candidate with greatest total score wins.

Nonetheless, no matter the system, a voting scheme must be secure enough to allow the anonymous expression of will. Shamos [1993] enumerates six commandments every electronic voting system should comply, they are cited in decreasing order of importance.

I Thou shalt keep each voter's choices an inviolable secret.

II Thou shalt allow each eligible voter to vote only once, and only for those offices for which she is authorized to cast a vote.

III Thou shalt not permit tampering with thy voting system, nor the exchange of gold for votes.

IV Thou shalt report all votes accurately.

V Thy voting system shall remain operable throughout each election.

VI Thou shalt keep an audit trail to detect sins against Commandments II-IV, but thy audit trail shall not violate Commandment I.

Any violation of the first three commandments must not be tolerated, it hardly can be considered an election if these principles are not respected. Commandment IV tends to be more flexible, depending on the jurisdiction some countries may allow some minor errors (e.g: $\pm$ 3 votes) in vote counting. To the best of our knowledge, commandments V and VI are not enforced strictly in most elections. These commandments can be translated to more specific properties that voting schemes are advised to provide.

Democracy The democracy property, also called *Elegibility*, states that only electors in the public census are allowed to cast a vote.

Uniqueness Electors can only vote once, double voting is not allowed. This property is usually implied in the Democracy one, since no election can be considered democratic if someone votes more than once.

Privacy It is not possible to relate a vote with the elector who casted it.

Verifiability Implies the existence of auditing mechanisms for the election, ensuring that the voting process has been correctly developed. We can distinguish three types of verifiability:

- Casted-as-intended: the ballot is sent with the desired vote direction.
- Recorded-as-casted: the ballot is recorded as it was sent.
- Tallied-as-recorded: the ballot will be tallied with the same vote direction as recorded.

When a voting scheme achieves the three of them, we can say it is end-to-end verifiable Benaloh et al. [2015]. End-to-end (E2E) verifiability is crucial property in electronic voting, because as we transition from paper-based elections, we lose the physical evidence, i.e. paper ballots, that accompanies votes, which makes detecting errors harder.

E2E verifiability places powerful auditing resources in the hand of electors. Nonetheless, it is a useless property if electors do not take and active role challenging the election process. Electors should challenge the system during the voting itself (casted-as-intended) and after the voting process (recorded-as-casted and tallied-as-casted). If a sufficient number of electors take advantage of E2E verifiability, they act as random audits that provide sufficient confidence that the election outcome is correct.

Additionally, if the final tally can be computed and verified by anyone, part of not of the election process, we say the protocol it is **Universally Verifiable**.

Integrity It is unfeasible for any party in the system to modify a ballot without the forgery being detected.

Accuracy Also known as *correctness*, it ensures that the results of an election are accurate and reflect the actual preferences of the voters. This requires that no one can change anyone else's vote, that all valid votes are included in the final tally, and that no invalid vote will be included in the tally.

Robustness Robustness ensures that no coalition of electors and/or parties can disrupt the election process.

Coercion Resistance An elector cannot prove how he voted, or can lie about the direction of the vote. The voter coercion problem Juels et al. [2010], Wu et al. [2014] describes an scenario where the elector might be intimidated (or bribed) to vote in a certain direction. Coercion resistance, and the problem of vote selling, are also closely related to the concept of receipt-freeness. An election system is considered receipt-free if the elector has no means to produce a proof of the direction of his vote. Since coercion resistant schemes tend to present higher computational complexity times, many systems resort to an alternative solution: allowing the modification of the vote. By allowing electors to later change the direction of their vote, voter coercion, or bribing, lose effectivity. Since any receipt can be invalidated with a new vote.

Electronic voting (or e-voting), as opposed to traditional voting, enables a new myriad of possibilities that help to bring voting to better and more secure standards. Traditional elections require physical and in-person attendance, which might be difficult for some people. And, their integrity depends on multiple individuals and organizations that might fail, or maliciously sabotage, to provide a valid election. The verifiability of traditional elections based on paper is extremely limited. Even in an ideal scenario where not even the slightest error occurred, electors have no means to verify the final outcome by themselves. Electronic voting aims to provide a more efficient, accessible and verifiable elections. Where the honesty and integrity of the system relies on mathematics and cryptography instead of individuals or lobbies. Nonetheless, e-voting also brings new challenges. Mainly security: the system needs to be secure against new and more sophisticated attacks, and usability: technology should help electors and facilitate voting instead of making the process incomprehensible for most citizens.

In this chapter, we review our three proposals for electronic voting , alongside with some implementations, that tackle these challenges from different angles. The three proposals provide the security properties previously described, although under different scenarios and with varying assumptions. The three systems later detailed are:

1. A light voting scheme based on blind signatures that only requires 2 authorities to operate.

2. A blockchain based voting protocol focused in public verifiability and the engagement of usual contenders as authorities.

3. A voting protocol with post-quantum security properties.

While the three presented protocols allow for flexible vote encoding and potentially could support any kind of voting systems, we only consider single race and multi-candidate elections for the rest of the document.

For the rest of the document, we will employ the feminine third person singular **she/her** to refer to an individual elector engaging with the system, and the masculine third person singular **he/him**, or third person plural **they/them**, to refer to the party, or parties, that participate making the election possible. This is done, in the same vein that many works in the field, with the sole purpose of providing some clarity and distinctly stating the agents in the system.

4.1 State of the Art

In this section, we review the most relevant works of electronic voting in the literature. Some of the works here described, are reviewed because of its importance in the field of electronic voting, others because the similarity in their goals, and some of them because of the similarity of the cryptographic primitives employed. We focus on cryptographic electronic voting systems. Cryptography-free secure systems Rivest [2006], Rivest and Smith [2007] and cryptographic paper-based systems Chaum [2004], Naor and Shamir [1994], Chaum et al. [2005], Ryan [2005], Bismark et al. [2009], Chaum et al. [2008], Essex et al. [2007],
Adida and Rivest [2006] are not the focus of this document, and therefore not covered unless specifically necessary. For further information on the history of elections and voting technologies we refer the reader to Saltman [2006].

Electronic voting publications can be categorized by the cryptographic primitives used to provide privacy while preserving the verifiability of the election. We can identify systems based in blind signatures, mixnets, ring signatures, zero-knowledge proofs, homomorphic cryptography, and blockchain. These primitives are detailed in Chapters 2 and 3.

The use of mixnets in an anonymous channel within mail services was the foundation for the earliest electronic voting scheme proposed Chaum [1981], Carroll and Grosu [2009]. What made this scheme remarkable was that it did not rely on a trusted authority, instead requiring only that at least one party remains honest throughout the cascade of mixes in order for the system to remain private. Despite the historical relevance of mixnets in e-voting, their vulnerabilities to correlation attacks Shmatikov and Wang [2006], and the need for large architectures with many parties have made them an archaic system for electronic voting. Nevertheless, there are many applications based on mixnets, such as the Tor browser, that make use of their privacy properties with remarkable success.

4.1.1 Blind Signatures

Blind signatures (see Section 2.6.1) introduce a method for anonymous signing. Election authorities can sign ballots from eligible voters, without compromising the privacy of the vote, and removing any possible link between the signed vote and the elector's identity.

Juang et al. [2002] present a robust and verifiable multi-authority system, which enables abstention after the registration phase and allows objections to the tally without compromising the privacy of the voter. The scheme utilizes distributed blind signatures Chaum [1983], Camenisch et al. [1994] to distribute the power of a single authority. The authors propose a complex architecture involving multiple partners such as electors, administrators, scrutineers, and a counter. The electors first encrypt their votes and apply blind threshold signature techniques to obtain their votes signed by administrators. In the voting phase, electors can generate their actual encrypted votes from the blind encrypted ones, and then send the vote to the counter through an untraceable electronic mail system. After the voting phase, the votes are published, and if there are no objections, the scrutineers send their pieces of the secret key to the counter. The votes are decrypted, and the results are published. The system ensures the privacy of electors from other entities of the system. The time complexity of the system depends on the complexity of the blind threshold signature scheme and the preparation phase where authorities need to cooperate.

Li et al. [2009a] proposed a multi-authority voting system, which is based on blind signatures. The system allows electors to cast their votes and blind them before getting them signed from multiple authorities. The authors suggest that the authorities should be made up of multiple parties from across the political spectrum to ensure honest functioning of the system. The scheme involves four voting phases, four authorities, three pairs of keys for each authority, and two pairs of keys for each vote. However, the high number of authorities and the processing of each vote result in a large number of modular exponentiations required. Additionally, some aspects of the public key infrastructure and the blind signature functions are not fully elaborated upon in the paper.

Thi and Dang [2013] proposed a novel election scheme based on blind signatures and dynamic ballots. The registration process for an elector is conducted through a chain of authorities, similar to the scheme proposed by Li et al. [2009a]. To safeguard the privacy of votes from coercion, dynamic ballots are employed, which change for each elector. The Ballot Center provides a random permutation of candidates to each voter. In the tallying phase, Plain-text-equivalence (PET) Jakobsson and Juels [2000] is used to eliminate invalid and duplicate ballots.

Aziz [2019] proposed an electronic voting system based on blind signatures. The system is designed to be multi-authority and coercion-resistant, where fake credentials Juels et al. [2010] are used to provide the elector with an exit mechanism in case of coercion. The system's registration process begins with the elector anonymously requesting a token from a token authority after which the ballot is constructed using the parameters in the token. Blind signatures are employed to sign the ballot from a registrar, and the signed ballot is cast through a mix-net. After the election, a set of trustees cooperates to decrypt the votes and compute the final tally. Ballots are shuffled and separated to prevent coercion, which also avoids the elector from knowing if their vote was tallied as cast. To disseminate trust, the scheme employs a distributed key generation protocol between a registrar and a set of trustees.

4.1.2 Ring Signatures

Ring signatures (see Section 3.3.2) present a way for electors to identify themselves as eligible members of the census, without revealing their identities. As long as the size of the ring is large enough, electors get sufficient statistical privacy.

Chen et al. [2008] propose an electronic voting system based on modified linkable ring signatures, where only designated verifiers can verify the validity of signatures using their private key. To do so, ring signatures are encrypted with the verifier's public key. This adds an extra layer of security to the system and prevents designated verifiers from broadcasting information about a private signature. The election's private key is generated and distributed among a group of tallying authorities using a (k, l)-threshold sharing scheme Gennaro et al. [2007], and the public key is made public before the election starts. The elector selects the direction of her vote, encrypts it using the public key of the election, and sends it anonymously to an administrator. The administrator signs the ballot, adds a timestamp, and returns the ballot to the elector. Then, the elector crafts the linked ring signature, encrypts it with the public key of a tallying authority, and publishes it on the bulletin board. After the voting phase, a minimum if k tallying authorities cooperate to recover the election private key, and check the validity of the received ring signatures to decrypt and publish the votes.

Salazar et al. [2010] proposed a novel voting system that leverages short-linkable ring signatures Tsang and Wei [2004]. The use of short-linkable signatures allows for the inclusion of a linking tag, which facilitates the association of votes from the same elector without compromising their anonymity. This important feature effectively mitigates the issue of double-voting while ensuring the privacy of the elector. The primary objective of their system is to minimize the involvement of third-party entities in the election process. In their approach, only a certification authority, responsible for key issuance and certificate validation, and a recount authority are required to ensure a fair and trustworthy election. The use of ring signatures enables signer ambiguity, while the linking tags serve as receipts that facilitate the removal of duplicate votes. Importantly, the system only requires the public key of the recount authority for encryption. However, this also means that a single authority has significant control over the voting process. The proposed system was later implemented in Tornos et al. [2014], where a modular implementation was provided in a multi-platform environment. Specifically, a desktop client, an Android native app, and a Firefox extension were developed to improve usability and engagement in the voting process.

4.1.3 Homomorphic Cryptography

Homomorphic cryptography Moore et al. [2014] enables a form of encryption that allows to perform computations over ciphertexts, without having to decrypt them, and obtain the same results as if we applied the computations over plaintext. Let p_1, p_2 denote two plaintext messages, an let $\mathcal{E}(p_1), \mathcal{E}(p_2)$ represent their encryption under some cryptographic scheme. Given two operations $\odot, \oplus$, a scheme is said to be homomorphic if, and only if, this equivalence exists:

$$p_1 \oplus p_2 = \mathcal{E}(p_1) \odot \mathcal{E}(p_2) \tag{4.1}$$

Most systems, such as RSA, present partial homomorphic properties. Meaning that the encryptions system only supports some operations under which the previous equivalence holds. Fully Homomorphic Encryption (FHE) Acar et al. [2018], allows boundless homomorphic computation over all possible operands supported by the scheme. FHE is the strongest definition of homomorphic cryptography, but it is based on other primitives that do require expensive relinearization and refreshment processes. For e-voting, and for the rest of this thesis, when using homomorphic encryption we refer to *partial* homomorphic systems, unless specified otherwise.

Homomorphic cryptography enables the aggregation of votes in such a way that the identity of individual electors gets diluted. Only the aggregated sum of votes is decrypted, preserving the privacy about the direction of individual votes.

Cramer et al. [1997] proposed a novel multi-authority voting system that relies on zero-knowledge and the homomorphic properties of the ElGamal cryptosystem ElGamal [1985]. The system employs a threshold scheme Desmedt [1994] to distribute the decryption private key among n authorities. In this approach, each authority publicly commits to sharing its secret in order to prevent malicious alterations. At least $\frac{n}{2}+1$ authorities are required to recover the secret key, which is used for the decryption of votes. The cost of the proposed scheme is linear with respect to the number of electors in 'Yes/No' elections. However, in the case of a multi-way election, the number of zero-knowledge proofs needed increases, resulting in a higher cost for the scheme.

Baudron et al. [2001] developed a multi-candidate and multi-authority voting system that employs homomorphic properties, zero-knowledge proofs and a threshold system similar to the one presented by Cramer et al. [1997]. The authors claim that their scheme provides receipt-freeness Benaloh and Tuinstra [1994]. The presented scheme uses the Paillier cryptosystem Paillier [1999] and is designed for a large group of electors organized hierarchically under authorities. In this system, votes move up in the hierarchy until their final decryption. The size of the vote depends on the number of zero-knowledge proofs, the size of hashed commitments, and the size of the modulus used in the Paillier system. The proposed scheme offers an improvement over previous systems by providing receipt-freeness, thereby enhancing the privacy and security of the voting process. Additionally, the use of a hierarchical organization of authorities allows for greater scalability in larger elections.

Porkodi et al. [2011] presented a voting scheme based on elliptic curves (see Section 2.1.4) and homomorphic cryptography. In the system, the elector encrypts her vote and posts it on a public bulletin board. Next, due to the homomorphic properties of the encryption, an encrypted tally can be computed anonymously from the bulletin board, thereby implicitly hiding the direction of the votes. Finally, the tally is decrypted in the final stage. The decryption secret key is shared between the authorities using a threshold scheme. Each authority has to prove that they posted a commitment to their private share, and each elector has to prove that they encrypted a valid vote. After this, the final tally is computed and published. However, each vote must be accompanied by an expensive zero-knowledge proof, and each authority must prove the validity of their commitment. This approach ensures the privacy and integrity of the vote, but it also increases the computational cost of the scheme due to the necessary zero-knowledge proofs.

4.1.4 Zero-Knowledge Proofs

Zero-knowledge proofs (see Section 2.4) present a powerful framework for e-voting thanks to its privacy properties. The most common approaches using zero-knowledge proofs usually involve either an elector crafting offline a proof to prove her entitlement to participate in the election, or the codification of the vote using an homomorphic scheme, and a prove that ensures the vote is properly formatted.

Cramer et al. [1996] proposed a novel multi-authority voting system. This system involves electors casting, encrypting, and distributing shares of their votes, which are then posted on a public bulletin board. Unlike other approaches that rely on expensive zero-knowledge proofs, they propose a non-interactive proof of validity that reduces the cost of the zero-knowledge proof from quadratic to linear. Each vote on the bulletin board is accompanied by one of these proofs to ensure its validity. To prevent a single authority from decrypting the vote, multiple authorities are used in a threshold system, which is similar to the one proposed by Cramer et al. [1997]. This multi-authority system is suitable for 'Yes/No' elections and is compatible with plurality voting, with minimal overhead. However, for other types of votes, the number of proofs required may increase significantly. The computational cost of distributing the commitments and checking the shares is linear for each partner involved in the elections. Specifically, voters require a linear effort in relation to the number of authorities and the size of the security parameter, while authorities require a linear effort in relation to the number of voters and the size of the security parameter.

Philip et al. [2011] proposed a receipt-free multi-authority system that takes advantage of the homomorphic features of the ElGamal cryptosystem. The system is designed for use with n authorities structured in a threshold scheme, along with a Trusted Center (TC). Prior to the election, the TC receives identification information from eligible electors and provides them with a username and pass code. Electors then use this information to register their vote and verify their eligibility. After each elector has encrypted and signed their vote, and crafted a zero-knowledge proof of correctness, the authorities use the distributed secret key to compute the final tally. To ensure receipt-freeness, votes are re-encrypted through the voting process. The authors provide a functional system with receipt-freeness, but note that this comes at the cost of increased computational complexity.

Yang et al. [2017, 2018] propose a ranked voting system where each ballot is represented as a square matrix. Each element of the matrix is independently encrypted using the ElGamal cryptosystem. To prove the correctness of the ballot, the elector must provide a proof of partial knowledge for every encrypted cell of the matrix, along with a zero-knowledge proof for the whole ballot. The homomorphic properties of the ElGamal cryptosystem allow for the combination of the rows of the ballots to compute the final tally for each candidate However, the decryption process requires the cooperation of all authorities. An improvement on the initial version of the system involves encoding the ranks of each candidate in binary format. This reduces the size of the matrix representing the ballot, and hence reduces the time complexity of the system. However, the main drawback of this approach remains the time complexity due to the large number of zero-knowledge and partial-knowledge proofs and modular exponentiations required. Despite this drawback, the proposed system provides a solution for ranked voting that ensures the correctness and privacy of each ballot. It is worth noting that this system is particularly suitable for small-scale elections, where the number of voters is limited and the time complexity is manageable.

4.1.5 Blockchain

Blockchain-based systems have become very popular as blockchain technology matured from a decentralized transactional system to a general computation and consensus layer. Its positive effect on transparency and voter confidence issues Moura and Gomes [2017] also helped to bring blockchain into electronic voting. Nonetheless, blockchain technology it is not risk free as we covered in Section 3.5. In fact, it can potentially worsen election security concerns Park et al. [2021] if integrated without proper caution. Introducing blockchain to the election technology stack might result in new attack vectors, as malicious parties might attack the consensus layer, infrastructure, or smart contract implementation, rather than the election itself. Additionally, this incorporation may lead to more complex voting processes and critical key management issues for users. Despite these trade-offs, blockchain-based electronic voting systems are nowadays fairly popular.

Works in the literature differ in how they employ the blockchain technology. We classify works in two categories: those that employ the blockchain as a decentralized public bulletin, and those works that employ smart contracts. We also differentiate between the systems that require to run a private blockchain, or a private set of nodes, in order to be fully operational. We refer the interested reader to these surveys to learn more about the challenges of blockchain systems Taş and Tanrıöver [2020], and their implementations Curran, Kshetri and Voas [2018].

Blockchain as a Public Bulletin Board

Require Private Blockchain: Ayed [2017] proposed a blockchain-based voting protocol in which each candidate has its own blockchain, and each block, except the first one, represents a vote for the candidate. The first block of each chain contains information about the candidate. To vote, an elector must first identify herself as a valid elector using her address and some private information. Then, she can decide the direction of her vote. When the vote is decided, some private elector's information is hashed alongside the hash of the previous block to create the hash of the new block. Finally, the new block is added to the corresponding candidate blockchain. While Ayed's proposal addresses some issues of centralized voting systems, such as decentralization and transparency, it fails to properly define some crucial parts of the protocol. Registration is not supported, and identification is assumed to be solved. Also, the encryption and identification processes are managed by a centralized interface, which raises concerns about the security and privacy of the system.

Hardwick et al. [2018] propose a blockchain-based voting system. Their registration phase involves blind signatures and relies on an honest authority to identify the electors. Unlike some blockchain voting systems, they do not employ smart contracts and require manual verification and tallying of votes by the electors. They also require involvement in block production and operate in private blockchains.

Do not require a private Blockchain: Noizat [2015] presents a voting system that utilizes blockchain and Merkle trees to ensure secure and accurate vote counting. The system employs a triple-key approach, where each elector uses public keys from the candidate she has chosen to vote for (KeyC), the election organizers (KeyA), and the voting application (KeyB). To maintain privacy, the system utilizes 2-of-3 multi-signatures Ruffing and Moreno-Sanchez [2017], which provide an added layer of security for the electors. The elector prepares a 2-of-3 multi-signature transaction to craft the ballot. It is worth noting that it is not possible to identify the elector or the candidate from a multi-signature address without having knowledge of all three public keys and who they belong to. To ensure the vote is counted as intended, the elector can verify the ballot using block explorers to confirm that their vote has been confirmed.

Lee et al. [2016] proposed a blockchain-based voting system that provides a secure and transparent approach to voting. The system requires an elector to send a transaction to the candidate's address to cast her vote. However, to vote, the elector must first register as a valid elector, which involves a two-stage registration process to maintain anonymity. To ensure anonymity, the registration process is split between two organizations, a registration organization, and a trusted third party (TTP). The elector sends a hash of her secret to both organizations to register, with the TTP verifying the elector's eligibility to protect her identity from the registration organization. Once the TTP confirms that the elector is on the census, the elector can send the transaction to vote for the candidate of her choice. To prevent multiple voting and ensure the validity of the elector's list, transactions are inspected and checked against a permutation of the TTP verified elector's list. This step ensures that unverified electors and those who have cast multiple votes are removed from the tally. The proposed blockchain-based voting system provides an efficient and secure solution for elections, ensuring transparency and accuracy. By utilizing a two-stage registration process and TTP verification, the system maintains the anonymity of electors, thereby preserving the integrity of the voting process.

Tarasov and Tewari [2017] present a voting protocol based on Zcash. Zcash, which emerged as a fork of Bitcoin, prioritizes privacy. This cryptocurrency supports two types of addresses: t-addresses, which operate like standard pseudonymous Bitcoin addresses and enable transparent transactions, and z-addresses, which maintain the anonymity and privacy of transactions. To achieve private-anonymous transactions, Zcash leverages special zero-knowledge proofs known as zk-SNARKS (see Section 2.4). These proofs enable the secure exchange of the secret values required to establish a private transaction between the sender and receiver. After registering, an elector can execute a transaction to the z-address of the desired candidate. The elector must also provide a valid t-address. As a result, the direction of the vote remains private, but the transaction itself is publicly visible. At the conclusion of the voting phase, candidates are expected to send all of the vote tokens to a central pool, where the final tally is calculated. It is important to note that this process relies on trust in the system and the candidates, as a malicious candidate could potentially interfere with the voting process.

Yang et al. [2020] proposed a blockchain-based voting protocol for range voting, in which each candidate receives a score, and the candidate with the highest score wins the election. To maintain the elector's privacy, the authors proposed a novel encryption scheme based on El Gamal and group-based encryption. Each vote is encrypted using the public key of the elector and the public keys of the candidates. This way, even at the end of the election, when the candidates release their secret keys, the individual votes remain secret. The authors take advantage of the homomorphic properties of El Gamal to compute the final score without decrypting individual votes. Then, the final score is decrypted by the candidates. Each transaction contains a vote, which consists of a set of scores, one for each candidate. However, each individual score needs to be double-encrypted, accompanied by a zero-knowledge proof and a partial-knowledge proof to ensure that it is a valid score from a valid voter. This requirement affects both the time and spatial complexity of the system. Despite this drawback, the authors provide a performance analysis that proves the validity of their election scheme. The proposed blockchain-based voting protocol ensures the privacy of individual votes and provides a secure and transparent approach to range voting.

Gao et al. [2019] propose an e-voting protocol based on blockchain technology that is resistant to quantum computing attacks. The authors achieve this by using code-based cryptography Niederreiter [1985], which is an NP-complete problem, instead of relying on traditional public key cryptography based on the difficulty of number theory. The protocol is also equipped with an audit function Al-Riyami and Paterson [2003], Hua-jie et al. [2014], which is based on public key certificates and enables the detection of fraudulent voters. To handle the certificates, the authors introduce the concept of a regulator, who does not participate in the election but has the authority to revoke voter privacy on demand. The authors provide a detailed computational time analysis of their protocol to demonstrate its feasibility. Overall, they present a promising approach to developing a secure and quantum-resistant e-voting system based on blockchain technology.

Wu [2017] presents a novel voting system that relies on ring signatures (see Section 2.6.2) and Bitcoin's blockchain. The protocol comprises two authorities: a Registration Authority (RA) and an Election Authority (EA). hese authorities are assumed to be trustworthy and are expected not to share information. Before the election begins, the EA generates and manages a public pool of Bitcoin addresses. Since Bitcoin provides anonymity rather than privacy, a two-phase registration process is carried out by the RA. This process decouples the elector's identity from the Bitcoin address and preserves the elector's privacy. Once the registration process is complete, the elector's public key is marked as valid. To craft a ballot, the elector must perform a ring signature, using her private key and a list of public keys, on her desired vote. This produces a ambiguous signature that makes it impossible to link the vote to her public key. After crafting the ballot, the elector selects a Bitcoin address from the pre-computed pool of addresses. The EA provides the elector with the associated private key to that address. The elector can then send the ballot as a transaction from her own address to the EA's address, with the ballot encoded in the transaction itself. The EA is responsible for retrieving all the ballots and verifying the integrity of the ring signatures to compute the final tally. Once the election is over, the list of public keys of electors is made public, allowing anyone to verify the correctness of the tally. However, since the ring signature used in this method allows for double voting, the EA checks the transactions coming from the same address, and only the latest transaction will be considered valid. It is important to note that the authorities wield great power in the voting process, which is a drawback of this method. Consequently, one of the requirements of the protocol is that the authorities comply with the desired conduct. Wu also developed a complete implementation of the system on Bitcoin's testnet, which includes a detailed definition of classes and use cases.

In their work presented by Cruz and Kaji [2017], an e-voting system that employs blind signatures on the Bitcoin network is proposed. The protocol involves three main entities: electors, an administrator, and a counter entity. To cast their votes, electors must first interact with the administrator to obtain a blind signature of their encrypted vote. After obtaining the blind signature, they unmask it and send the unmasked vote to the counter. When the voting period ends, the counter verifies the signatures and decrypts the received votes. The Bitcoin blockchain is used as a public bulletin board by both the administrator and the counter. The administrator publishes the identification and masked votes of the electors, while the counter publishes the signed votes and addresses used to cast the votes. This approach provides privacy and anonymity for the electors, as their identities and votes remain hidden. However, the trustworthiness of the administrator and the counter is critical to the security and integrity of the system. Furthermore, the use of Bitcoin as a public bulletin board may limit the scalability of the system due to the network's limited transaction processing capabilities.

Smart contract based elections

Hjalmarsson et al. [2018] propose a smart contract based voting scheme. In their system, two entities are required to run the election: electors, and election administration officers. To obtain a unique wallet, electors need to go trough an identification process. Solely accredited wallets are able to send a valid vote. In order to cast their votes, electors must communicate through ballot smart contracts which are dependent on the district. Verification of these votes is performed by a Proof-of-Authority (PoA) network that is external to the blockchain and also run by the election administrators. If the verification process succeeds, the transaction containing the vote is added to the blockchain by the PoA network. However, this approach has two main limitations. Firstly, it requires a dedicated PoA network to scan the blockchain, which can be a resource-intensive task. Secondly, the election administrators have complete control over the system, including the creation of the election, the privacy of users, and the validity of the votes. As a result, the electors' privacy is not fully guaranteed, and there is no proper distribution of responsibilities. In other words, the election administrators hold all the power, which undermines the democratic principles of transparency and fairness.

Lai and Wu [2018] introduce an elegant voting system that leverages Ethereum smart contracts and one-time ring signatures. In this system, each transaction represents a vote, and electors can cast their votes by making a transaction to their preferred candidate. The privacy of the elector is protected by ring signatures, which prevent double voting while also keeping the identity of the elector anonymous. To ensure fairness and prevent information leakage until the tally phase, the electors are advised to consider the stealth addresses of the candidates when casting their vote. This way, until the stealth addresses are revealed, the votes remain confidential. To reveal the stealth addresses, key managers are required. These key managers share the first private key of a candidate through a Diffie-Hellman interchange. Additionally, key managers are required to store some ETH in a deposit prior to the election. To recover their deposit, they must open their secret during the tally phase. Once the stealth addresses are revealed, the entire election process becomes public and can be audited on the Ethereum blockchain. All the requirements of the protocol are encapsulated within a smart contract, ensuring transparency and trust in the election process. This system offers a more democratic approach to voting, where the privacy of the electors is protected while also ensuring fairness and transparency.

Chouhan and Arora [2022] propose a blockchain-based election scheme that is built on the Hyperledger[1] blockchain framework. This implementation is designed to be compatible with most election setups and supports an unlimited number of electors. To maintain the privacy of the election results until the tallying phase, the authors use Shamir's secret sharing scheme (see Section 2.2). In this scheme, votes are encoded as points on a polynomial that is distributed among a set of trusted authorities. At the end of the voting phase, these authorities interpolate the polynomial to recover the vote. To ensure the anonymity of the electors, their identities are mapped to anonymous identifications during the registration phase. However, it is important to note that this approach is only possible because Hyperledger is not completely transparent and only produces permissioned distributed networks, not truly decentralized ones. The authors provide a detailed explanation of the Hyperledger contracts used in the implementation. However, the code is not open-sourced, which may limit the ability of other researchers to verify and reproduce the results.

Onur and Yurdakul [2022] propose a smart-contract based election scheme for ranked voting that employs Zero-Knowledge proofs for privacy. Their protocol is developed in Solidity for Ethereum compatible chains, and they provide an open-source implementation. During the registration phase, electors produce a commitment of their identity that is stored in a Merkle tree Merkle [1987]. Later on, during the voting phase, they can create a zero-knowledge proof that demonstrates they are eligible electors in the census without revealing where their commitment is stored in the Merkle tree. To keep the vote secret until the tallying phase, a commit and reveal scheme is used. The authors acknowledge that general zero-knowledge proof systems are computationally intensive, limiting the number of potential electors by the size of the Merkle tree. Increasing the size of the Merkle tree in a zero-knowledge proof system can have a significant impact on the computational resources required to generate the proof. This is because the proof generation process involves performing computations that depend on the height of the Merkle tree, which can be computationally intensive. Therefore, while increasing the size of the Merkle tree can allow for more potential electors to participate in the election, it may also make the proof generation process slower and more resource-intensive. Balancing the size of the Merkle tree

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

with the computational resources available is an important consideration for this election system.

4.2 TAVS: A two Authorities Voting scheme

We present a verifiable voting scheme that, if some conditions are met, guarantees that the desired properties of an electronic voting system, as described in Section 4, are provided. Particularly, guarantees that an individual voter's identity and their choice of candidate cannot be linked (privacy);that no party can modify a ballot without detection (integrity); that only verified and accurate ballots are included in the final tally (correctness) and any elector in the census can confirm that their vote has been accounted for accurately (verifiability).

The Two-Authorities Voting Scheme (TAVS) is a novel protocol, which can be viewed as a modification of blind signatures (discussed in Section 2.6.1) for electronic voting. TAVS is built on the foundation of two separate and unrelated authorities: an Identification Authority (IA), which verifies the eligibility of voters by checking their membership in the census; and a Remote Polling Station (RPS), which is where voters cast their ballots. The protocol's strength lies in its ability to achieve universal verifiability without relying on time-consuming primitives like zero-knowledge proofs. With TAVS, the electoral process is made more secure, transparent, and trustworthy, ensuring the integrity of the results.

Our goal is to introduce a more efficient and simpler voting scheme, where simpler regards to the number of parties implied in the election, as well as to the number and complexity of the steps to carry out. The outcome of these improvements, imply a reduction of the overall time-complexity. The simplicity of TAVS also allows for a better understanding on how the system works. Which enables voters to get a better understanding of the scheme despite not being experts on the field. TAVS allows the elector to verify that her vote has been included into the final outcome of the election. The final tally can be also later audited by any interested party, in order to verify the correction in the count. Privacy is guaranteed in all the election processes.

In order to allow the anonymous certification of the elector's vote, without compromising her privacy, we make use of a blind signatures (see Section 2.6.1). We present the scheme taking into account the RSA signature method (see Section 2.3.2) because of the homomorphic properties of the modular exponentiation. Nevertheless, the protocol here proposed can be modified to consider any other scheme with homomorphic properties.

As mentioned above, it is assumed that the two authorities implied are not related in any way. Also, unlike Chaum's original proposal of blind signatures, no communication is established in order to share elector's information. It is not assumed the authorities are honest and provide methods to detect malicious behavior of the authorities. From now, and during the formal description of TAVS, we assume:

- We do not enforce any specific way to encode the votes and only takes into account the numeric value resulting from the binary representation of the vote, regardless of how it is obtained. Given this flexibility, TAVS is compatible with many voting modalities (as presented in this Chapter introduction), as well as single or multiple races.

- TAVS is built upon the assumption that all auxiliary methods and procedures function correctly and that no weakness in the scheme can be derived from them. Specifically, the signature and hash functions are assumed to be secure.

- We assume that the organization responsible for conducting the elections provides each eligible voter in the census with a private identification. This identification is essential for verifying a voter's eligibility to participate in the election. Depending on the specific features of the census and its geographical distribution, the identification can be implemented either physically or electronically and distributed in various ways before the election date. Moreover, any existing suitable identifier can be considered as well

- It is assumed that the communication channels used for the electoral process are secure. This is critical to ensure that the electoral process remains trustworthy and free from interference or manipulation. It is important to note that the distributed identification provided to each eligible voter in the census can be used to implement secure communication channels between the voters and the authorities involved in the voting scheme during the elections. These secure channels can help ensure the confidentiality and integrity of the communication, preventing unauthorized access or tampering of the messages exchanged between the authorities and the voters.

4.2.1 Description of our Proposal

Here we describe our voting protocol, TAVS, based on multiple authorities and blind signatures. Unlike most of the systems based in blind signatures described in the Section 4.1, we reduce the number of required authorities to only 2. Thus, we simplify the voting procedure to just three steps and reduce the overall time computational cost. Contrary to many e-voting protocols, the corruption of a single authority does not compromise the security of the voting protocol, and the elector is able to check the honesty of the authorities. As described in Section 4.2, TAVS uses RSA to implement blind signatures, and the computations involved in the signing scheme are fully disclosed. The blinding scheme is employed to address the identification of the elector, and potential re-blocking issues and disagreements with the authorities are also taken into account. Let us once again note that TAVS does not enforce any specific way to encode the votes, and any protocol implementing the proposed scheme must clearly state the way the vote is coded. By providing this flexibility, TAVS can be adapted to various voting modalities and election scenarios.

TAVS consists on three sequential steps and is fully described in the following sections. In short, the first step consists on the generation of the pre-ballot. This process is carried out by the elector with no need to interact with any existing authority. The output of this process is an uncertified masked ballot (the pre-ballot) which cannot be disclosed without the elector compliance. The second step consists in the submission of the elector's identification and the (masked) pre-ballot to the IA in order to certificate the pre-ballot using an blind signature procedure. This certification is carried out whenever the identification corresponds to an elector in the census who has not previously ask for another vote to certificate. The certification process ends when the elector acknowledges safe and correct receipt. In the last step, the elector anonymously submits the certified ballot and the information needed to unmask the vote to the RPS. The generation of the pre-ballot is such that it is unfeasible the manipulation of the certified version of the ballot in order of obtain more than one valid votes.

In order to maintain privacy and provide democracy and verifiability, two public bulletin boards are used: the *Revoked Board* and the elections' *Public Bulletin Board*. The use of the *Revoked Board* prevent malicious electors from using certified ballots before the end of the certification process. The elections' *Public Bulletin Board* allows the electors to confirm that their ballot has been counted in the final tally.

Although the details will be provided in Section 4.2.1, we denote with T_H the size of the employed generic hash function, and with T_S the size of the signature key (number of bits in the binary representation of the modulus n), which will be important in the following because of their role in the coding of the vote. It is worth noting that regardless of the format in which the vote is presented, it will be encoded using a fixed number of bits, specifically $T_S - T_H - 1$. This approach ensures that the proposed scheme operates correctly regardless of the chosen encoding format. Therefore, the primary concern is to ensure that T_S and T_H are large enough to accommodate the encoding of the vote adequately. This provides the necessary flexibility to use different encoding schemes and ensures compatibility with a variety of voting modalities.

Pre-ballot generation

Before the voting process begins, it is necessary to agree on the methods to be used for hashing and electronic signature scheme. Thus, the IA is responsible for generating an RSA key, and broadcasting the public component of this key to all members in the census. This enables every member to verify the correct validation of the ballot. The RSA key generated by the IA consists of a private component $S_{IA} =< s >$ used for signing, and a public component $V_{IA} =< n, v >$ used for verification. These components allow for the certification of the ballots and for checking their correctness in the tallying process.

Once the public component of the IA signature key has been distributed, the elector can generate a pre-ballot with her vote. The elector can independently perform this process isolated from the other two authorities implied in the election. The procedure to generate the *pre-ballot*, a ballot that has been generated but not yet validated, is depicted in Algorithm 8 and it ensures the following properties:

- The pre-ballot must be concealed in such a way that no-one but the elector is able of finding out the direction of the vote. The elector is the only one that knows the mask that hides the votes.

- In order to prevent double voting, the mask must be linked to the vote so that it would be unfeasible the manipulation of the pre-ballot.

Algorithm 8 Pre-ballot generation.

```
Require: $IA$'s public component of the $RSA$ signature key $V_{IA} =< n, v >$
Require: A hash function $h$ of $T_H$ bits.
Ensure: A pre-ballot ready to be (blindly) signed by the $IA$
1: Let $T_S$ be the number of bits needed to encode $n$.
2: Let $vote$ be the $T_S - T_H - 1$ bits coding of the elector's choice
3: Let $1 < mask < n$ be a (private) randomly-generated value such that $gcd(mask, n) = 1$
4: Compute $mask^{-1} \bmod n$
5: Let $hash = h(vote \| mask)$
6: $preballot = (vote \| hash) \cdot (mask^v \bmod n) \bmod n$
7: return $< preballot,\ mask >$
```

First, the procedure encodes the direction of the elector's vote (line 2 in Algorithm 8), and selects a random generated mask that will be kept secret until the ballot is submitted to the RPS.

In order to prevent forgery, the concatenation of the elector's vote and the mask is hashed. These three elements (the elector's vote, the randomly generated mask and the hash) are combined to obtain the pre-ballot (line 6). In order to avoid possible reblocking errors in the certification due to a pre-ballot greater than n value, the size of the elector's vote codification is such that the concatenation of the vote and the computed hash results in a number lower than n. The structure of the pre-ballot is depicted in Figure 4.1.

We note that the computed hash is not explicitly included in the output of Algorithm 8 but that it is present in an implicit way. The secrecy of the mask is critical, and what holds the desired security properties of an electronic voting system. Once the elector crafts the pre-ballot and the mask, she interacts with the IA to certificate her ballot.

Elector identification and pre-ballot certification

The next stage in our system involves the blind certification of the pre-ballot by the IA. First, the elector sends the masked pre-ballot along with her identification to the IA. This process is outlined in Algorithm 9.

It is important to note that the IA is unable to determine the direction of

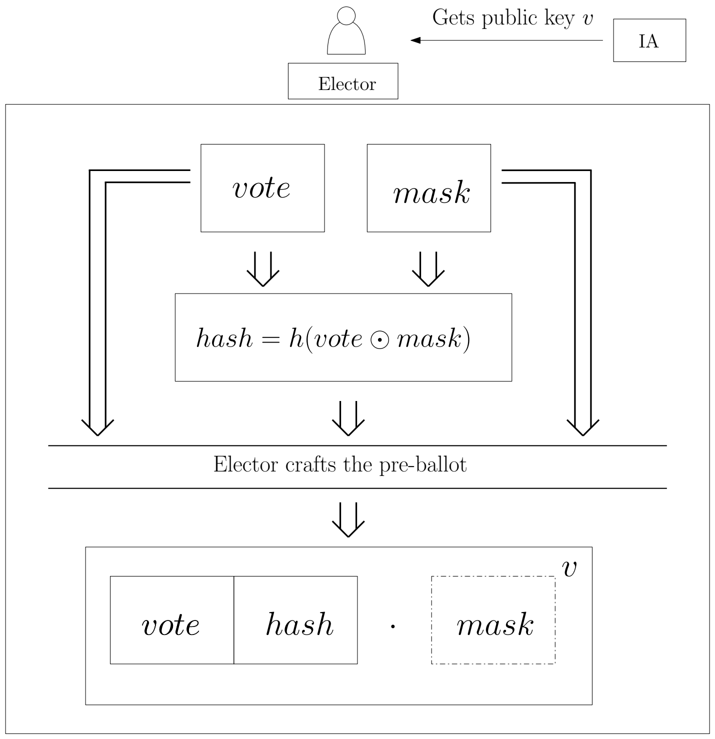


Figure 4.1: The pre-ballot structure is presented graphically with each box representing a string. The modular exponentiation operation is represented by a dashed box. Concatenation of boxes represents the concatenation of strings. This image depicts the organized structure and crafting of the pre-ballot that the elector will send to the IA, as a result of following the steps outlined in Algorithm 8).

the elector's vote as long as the mask remains undisclosed. Thus, the elector has complete control over her vote and may even choose to cast a blank or null vote. Upon receiving the masked pre-ballot and identification from the elector, the IA first checks that the elector has not requested another ballot for certification previously. If this condition is satisfied, the IA records the elector's identifier and proceeds to certify the pre-ballot by signing it using the publicly shared RSA signature key, as shown in line 7 of Algorithm 9.

We note the effect of the certification process on the mask. We denote with *pb* the pre-ballot built by the elector. Because of the homomorphic properties of the modular exponentiation in RSA it follows that:

Algorithm 9 Pre-ballot certification

```
Require: A pre-ballot $pb$ generated by an elector
Require: Elector's identification
Require: The public $V_{IA} =< n, v >$, and private $S_{IA} =< s >$ components
    of $IA$'s $RSA$ signature key
Ensure: A validated ballot (pre-ballot signed by the $IA$) or
          $Forgery\ attempt$
 1: if the elector's identifier is already stored in the $IA$ records then
 2:     return $Forgery\ attempt$
 3: end if
 4: Store the elector's identifier in the records
 5: $endCertification = False$
 6: while not $endCertification$ do
 7:     Compute $b = pb^s \bmod n$                    // Certified ballot
 8:     Store $b$ in the $Revocation\ Board$
 9:     Send $b$ to the elector and ask for validation
10:     if the elector's validates $b$ then
11:         Remove $b$ from the Revocation Board
12:         $endCertification = True$
13:     end if
14: end while
```

$$\begin{aligned} pb^s \bmod n &= ((vote\|hash) \cdot mask^v)^s \bmod n = \\ &= (vote\|hash)^s \cdot (mask^v)^s \bmod n = \\ &= (vote\|hash)^s \cdot mask \bmod n \end{aligned}$$

The process asks the elector to acknowledge the correct certification of the ballot, that, because of the structure of the pre-ballot, it only needs the elector to compute the inverse of the mask modulus n to unmask the ballot:

$$\begin{aligned} &((vote\|hash)^s \cdot mask \bmod n) \cdot mask^{-1} \bmod n = \\ &= ((vote\|hash)^s \cdot mask \cdot mask^{-1} \bmod n = \\ &= (vote\|hash)^s \bmod n \end{aligned}$$

afterwards, it is possible to use the public component of the IA signature key to verify that the certification considered the pre-ballot previously sent by the elector:

$$((vote\|hash)^s)^v \bmod n = vote\|hash.$$

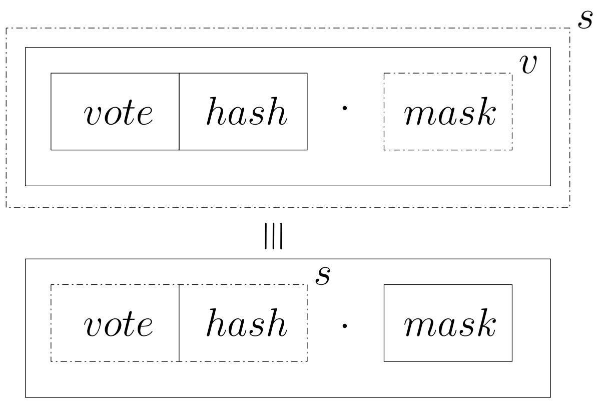


Figure 4.2: The image graphically represents the certification of a ballot as described in Algorithm 9. Once validated, the IA signs the pre-ballot using its secret key. Both structures are equivalent thanks to the homomorphic properties of the modular exponentiation. The same considerations explained in Figure 4.1 apply.

After the elector receives the certified ballot, she is required to acknowledge the receipt of the ballot. This is necessary to prevent a malicious elector from claiming that the ballot was non-correct later, which would enable double-voting. Once the acknowledgment is received, the IA stores the certified ballot in a board of revoked ballots, as shown in line 8. The ballot is removed from this board once the elector acknowledges the safe and correct receipt of the ballot. The complete process of certification and the structure of the signed ballot can be observed in Figure 4.2.

Submission and publication of the ballot

The third and last step in the voting scheme implies the submission of the certified vote to the RPS. The procedure is defined in Algorithm 10.

In short, for any certified ballot, an identical procedure to the one that allows the elector to verify the certification of her ballot, allows the RPS to access to the masked vote (lines 4 and 5 in Algorithm 10). Thus, the RPS can obtain the vote itself (line 6) and the hash that relates the vote and the mask (shown in line 7). The computation of $h(vote\|mask)$ allows the RPS to check the integrity of the ballot (shown in line 8). Please note that the certified ballot is not related in any way to the elector's identity and the vote cannot be related with her. The process is depicted in Figure 4.3.

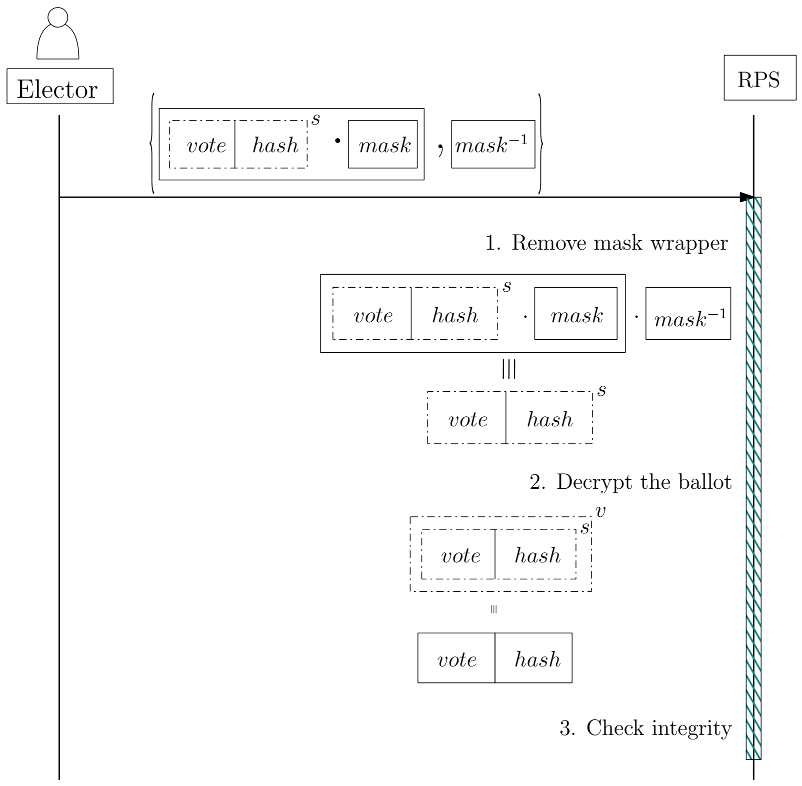


Figure 4.3: Sending the ballot to the *RPS*. When the *RPS* receives the ballot, and the inverse of the mask, it applies Algorithm 10 to recover the elector's vote and its associated hash. Both are published on the bulletin board.

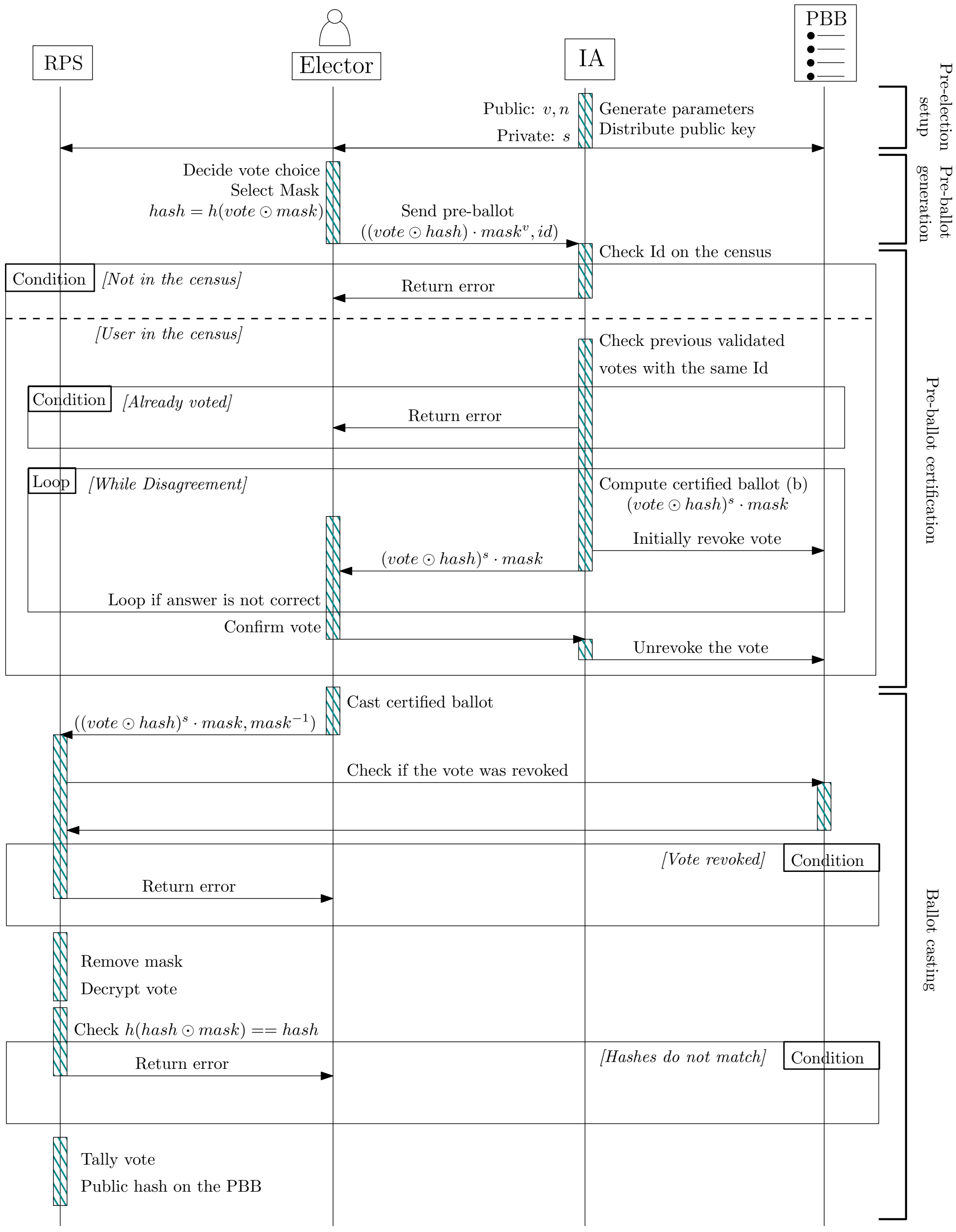


Figure 4.4: The timing and interaction diagram presented in the figure illustrates the entire process that an elector must follow in order to successfully cast a vote using the proposed voting scheme. The diagram shows the various computations and interactions required between the different parties involved in the process.

Algorithm 10 Ballot casting.

```
Require: A certified ballot $b$
Require: The mask used in the generation of the pre-ballot $mask$
Require: The $IA$ public component of the $RSA$ signature key
Require: The agreed hash function $h$
 1: if $b$ is in the $Revocation\ Board$ then
 2:     return $Forgery - attempt$
 3: end if
 4: Compute $pkg = b \cdot mask^{-1} \bmod n$
 5: Compute $pkg = pkg^{v} \bmod n$
 6: Set $vote$ equal to the first $T_S - T_H - 1$ bits of $pkg$
 7: Set $hash$ equal to the last $T_H$ bits of $pkg$
 8: if $hash == h(vote \| mask)$ then
 9:     Store the casted $vote$
10:     Publish $hash$ in the elections' $Public\ Bulletin\ Board$
11:     return $Correct$
12: else
13:     return $Forgery\ attempt$
14: end if
```

The assumptions we consider in the description of TAVS prevent the communication between both authorities. Nevertheless, the IA would be able to recover the direction of the vote of any elector if the public bulletin board includes some information he could link to an elector identifier. This is not possible because the hash is not available to the IA, and, therefore, its publication does not allow anyone to link the elector and her vote.

Figure 4.4 shows the complete voting process: the interactions between the partners; the timing in the voting scheme; and, the conditions that trigger different consequences as well. As it can be seen, the process is initiated by the IA by making public the parameters and the public key components. Then, the elector carries out Algorithm 8 and sends the pre-ballot to the IA. Assuming the elector is on the census and did not vote before, the IA responds the elector with a certified ballot which is considered invalid until the elector confirms the vote. The elector can independently check the validity of the received certified ballot and challenge the IA if it does not match her initial ballot. After the certification phase, the elector send her ballot to the RPS. There, it is checked if the vote was revoked by the IA.

If the vote is valid, RPS follows Algorithm 10 to check the vote is correctly crafted. If the algorithm outputs true, the vote is counted and its hash is posted on the public bulletin.

4.2.2 Properties of the voting scheme

This section focuses on the security analysis of TAVS and how it relates to the essential properties of voting schemes mentioned in the introduction of Section 4. To begin, we acknowledge the reliability of RSA and various hash functions available in the literature that are suitable for use in the TAVS. We also recognize that the safety of blind signatures and the limited information that authorities have access to are crucial factors in ensuring the security of TAVS.

Democracy

The scheme here proposed does not prevent malicious electors to access the RPS. Nevertheless, on the one hand, we note that it is not feasible for adversaries to construct a certified ballot unless they could gain access to the IA signature private key. On the other hand, in order to impersonate an elector it is necessary to know the identification of the elector, which is assumed to be private.

Uniqueness

Only one ballot per elector can be certified by the IA. We prove that it is not feasible to tamper with a certified ballot, nor to maliciously craft a pre-ballot in order to achieve double voting.

Let us recall that once a ballot is certified by the IA it is of the form $(vote\|hash)^s \cdot mask$, where $hash = h(vote\|mask)$. Obtaining another valid ballot from would imply:

- That $(vote\|hash)^s$ can be considered as a mask of a (tampered) $vote'$ coded into $mask$.

  Indeed, it is feasible that there will exist the inverse of $(vote\|hash)^s$ modulus n and therefore a valid mask for $vote'$. Nevertheless, taking into account the construction described in Algorithm 8, to tamper with the ballot it would also be necessary that $mask$ were such that:

$$mask = vote' \| hash',$$

where $hash' = h(vote' \| (vote \| hash)^s)$. This is highly unfeasible taking into account that the certification (by signature) can only be carried out by the IA and the result is unknown for any elector before the certification process. The hash acts like a redundancy parameter, making the search for manipulated masks unfeasible. This is related with the third property needed for blind signatures stated by Chaum (see Section 2.6.1). This proves the search for valid blind signatures, through tampered masks, to be extremely unfeasible, as it would imply the possibility of finding hash collisions at will.

- That the construction of the ballot considers a $mask$ that can be modified after the certification of the ballot in order to obtain a new (tampered) vote.

  In other words, any attempt to modify the mask used to construct a certified ballot will result in a different hash value, which in turn will produce a different ballot. Therefore, attempting to obtain another valid ballot by modifying the mask is highly unlikely, if not impossible. This demonstrates the robustness of the TAVS scheme against double voting attempts.

Taking into account the two points we just described, it is proved that the uniqueness of the voting is granted. No double voting or tampering is feasible.

Privacy

It is not possible to relate a vote with the elector who casted it. TAVS requires the elector to interact with two different authorities: the IA has access to the elector identifier but he cannot unmask the vote, as the mask is required to do so, and the RPS has access to the direction of the vote but the elector identifier is not sent to him. Therefore, it is impossible to relate an elector with the direction of her vote unless both authorities share information, which contradicts an assumption of the scheme.

Integrity

A similar reasoning as the one presented in the uniqueness property proves that TAVS preserves the integrity of the vote. The construction of the ballot described in Algorithm 8, allows to state that it is unfeasible for any partner to modify a previously certified vote.

Accuracy

TAVS guarantees that no invalid votes are tallied. We also note that the RPS has enough information to be audited by any party. Once again, without access to the IA's signature (private) key, it is unlikely to design a method able to substitute a set of valid certified votes by another set of forged ones.

Verifiability

Any elector in the census can verify that her vote has been taken into account in the way it was cast and included in the final tally. The participation of an audit authority would provide the electors the confidence their votes have been counted in the direction they were cast. We note that, because the unfeasibility of tampering with the ballots, any audit authority is able verify the tally using only the information stored in the RPS. Hence, TAVS also presents universal verifiability.

Coercion resistance

Given that there is a trade-off between coercion resistance and performance, we do not focus on this dichotomy. We provide a scheme where the elector has mechanisms to be certain that her vote was considered, as well as enough information for an independent audit authority to anonymously check the correctness of the tally.

4.2.3 Time complexity analysis

In this section, we shall examine the time complexity of TAVS. Our aim is to demonstrate that our system has a linear scaling behavior in relation to the number of votes. While it is common practice in the literature to use modulo n operations, such as addition, multiplication, and exponentiation, as the unit of computational cost, the cost of concatenation is not typically taken into consideration. However, we will analyze the time complexity in terms of bit operations, with a particular focus on the complexity of various operations. Throughout this section, we will use n to represent the number involved in these operations, and $\log n$ to denote the number of bits in n.

It is well known that modular addition and subtraction have time complexity of $\mathcal{O}(\log n)$ bit operations, multiplication and inverse have a complexity of $\mathcal{O}(\log^2 n)$ and modular integer exponentiation is the most expensive operation with $\mathcal{O}(\log^3 n)$ time complexity. We make use of a hash function, which we recommend it to be a recognized and established one. We assume the complexity of applying the hash function is linear with respect to the input.

Pre-ballot generation

Once the parameters of the election have been decided, the elector must follow Algorithm 8:

1. The elector must select a random mask in line 4 of Algorithm 8, the mask must be invertible. Since we require $gcd(mask, n) = 1$ the inverse exists. To find the inverse we suggest extended Euclid's algorithm Menezes et al. [1996] which has a complexity of $\mathcal{O}(\log^2 n)$ bit operations.

2. Applying the hash function of line 5 requires linear effort with respect the input. In this case, the input is the concatenation of the vote and the mask. This results in a complexity of $\mathcal{O}(\log n - T_H - 1 + \log n) = \mathcal{O}(\log n)$ bit operations.

3. To craft the pre-ballot in line 6 of the algorithm, a multiplication and a modular exponentiation are needed.

The time complexity of Algorithm 8 can be expressed as:

$$\mathcal{O}(\log^2 n) + \mathcal{O}(\log n) + \mathcal{O}(\log^2 n) + \mathcal{O}(\log^3 n) = \mathcal{O}(\log^3 n)$$

Observe that modular integer exponentiation determines the time complexity of the algorithm.

Pre-ballot certification

For Algorithm 9 we assume the best case: IA is benign, the channel is secure and the elector will validate the ballot. Computing the certified ballot requires:

1. One modular multiplication and one modular exponentiation are required in line 7 to sign the ballot.

2. To check if the received certified ballot is correct, the elector needs to perform one multiplication and one modular exponentiation in line 10. It is not explicitly shown on Algorithm 9 but it can be seen on Figure 4.4.

The time complexity of Algorithm 9 is $\mathcal{O}(\log^2 n) + \mathcal{O}(\log^3 n) = \mathcal{O}(\log^3 n)$ bit operations for the elector and $\mathcal{O}(\log^2 n) + \mathcal{O}(\log^3 n) = \mathcal{O}(\log^3 n)$ bit operations, per ballot, for the IA. Again, modular exponentiation determines the time complexity of the algorithm.

Ballot Casting

Once the elector sends the ballot to the RPS she does not need to perform more computations. The RPS takes part on Algorithm 10 and performs the following to decrypt the vote:

1. A multiplication is used in line 4 to remove the mask wrapper from the ballot.

2. A modular exponentiation is needed in line 5 to recover the vote.

3. To check the integrity of the ballot the hash function must be applied in line 8 with $\mathcal{O}(\log n - T_H - 1 + \log n) = \mathcal{O}(\log n)$ bit operations.

The time complexity of Algorithm 10 is $\mathcal{O}(\log^2 n) + \mathcal{O}(\log^3 n) + \mathcal{O}(\log n) = \mathcal{O}(\log^3 n)$ bit operations per processed ballot. As in the previous algorithms, the complexity is dominated by the modular exponentiation operation.

TAVS complexity

Here we summarize the total complexity of the system. We differentiate between the complexity required for the elector to cast a vote and the complexity for the system to undertake the whole election process. The elector must perform a total of $\mathcal{O}(\log^3 n)$ bit operations to vote, as shown in Algorithms 8 and 9. TAVS must carry out a total of $\mathcal{O}(\log^3 n)$ bit operations per processed vote (Algorithms 9 and 10).

The computation of modular exponentiations determines time complexity of our method. If this operation is considered as the atomic operation in the analysis of the complexity, as many works in the literature do, it is clear that the elector computes a constant number of exponentiations to vote. The same applies to the involved authorities: they compute a constant number of exponentiations per ballot. Therefore, the complexity of the system scales linearly with the number of votes to be processed. Since the complexity of the system does not depend on the number of candidates, the number of authorities involved, or the encoding of the vote, we can state that TAVS can be easily extended and scaled to adapt more general situations.

For an implementation of TAVS with smart contracts that covers the costs of running your own election, please see Appendix A.

Comparison with other systems

The purpose of this section is to compare the performance of TAVS with other works that share similar methods and objectives, as discussed in Section 4.1. It should be noted, however, that some of the works reviewed in the literature do not offer a comprehensive analysis of their system's time complexity or provide only partial details. In some cases, these works may consider a centralized authority, leaving decentralization as a future or theoretical exercise. Here we present a comparison of the theoretical time complexity of previous methods in the literature and TAVS.

In order to state the time complexity of the methods, we consider exclusively the number of modular exponentiations (the most expensive operation). The complexity is then expressed as an asymptotic function on the number of bit operations. When the methods do not specify the protocol used (e.g: which kind of zero knowledge proof was used) we introduce a new variable in the cost analysis. If possible, we group the different number of authorities involved as a single variable. Multi-candidate elections scenarios are considered. Table 4.1 shows the results of this comparison. We also differentiate between the costs assumed by the elector and the costs associated to the system to process a single vote.

	Elector's Cost	System's Cost per vote
Chaum [1981]	$\mathcal{O}(m\log^3 n)$	$\mathcal{O}(m\log^3 n)$
Cohen and Fischer [1985]	$\mathcal{O}(\log^5 n)$	$\mathcal{O}(\log^5 n)$
Cramer et al. [1997]	$\mathcal{O}(c\log^3 n)$	$\mathcal{O}(ac\log^3 n)$
Juang et al. [2002]	$\mathcal{O}(m\log^3 n)$	$\mathcal{O}(a\log^3 n)$
Baudron et al. [2001]	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(a\log^3 n)$
Cramer et al. [1996]	$\mathcal{O}(t\log^3 n))$	$\mathcal{O}(a\log^3 n))$
Li et al. [2009a]	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log^3 n)$
Philip et al. [2011]	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(a\log^3 n)$
Essex et al. [2012]	$\mathcal{O}(c\log^3 n)$	$\mathcal{O}(ac\log^3 n)$
Thi and Dang [2013]	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log^3 n)$
Yang et al. [2018]	$\mathcal{O}(c(\log P)\log^3 n)$	$\mathcal{O}(ac(\log P)\log^3 n)$
Aziz [2019]	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log^3 n)$
TAVS	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log^3 n)$

Table 4.1: Table representing the asymptotic cost of the work performed by the elector and the system in number of bit operations. In the table: m references the number of layers on a mix-net, a represents the number of authorities, c represents the number of candidates in the election, t is the number of different shares a vote is fragmented and P the number of points awarded in a ranked voting system.

We proceed to briefly explain how the values for Table 4.2 were computed. In Chaum [1981], an elector needs to encrypt, using RSA, the vote and re-encrypt it as many times as layers m has the mix-net. The cost for the voter depends on m, $\mathcal{O}(m(log^3 n)$. Each vote needs to be encrypted $m + 1$ times, shuffled and latter on, decrypted $m + 1$ times. The cost can be computed as $2\log^3 n(m + 1) \approx \mathcal{O}(m\log^3 n)$.

Cohen and Fischer [1985] report a 4 phase protocol. These phases include several modular exponentiations and nested iterations. The elector needs to carry out 2 of these phases while the authorities have to complete the other two. The authors expose that the total expected time required by both the authorities and the voter phases is $\mathcal{O}(\log^5 n)$. In Cramer et al. [1997], to cast a vote, the elector needs to perform an encryption of the vote using ElGamal cryptosystem and a proof of partial knowledge. This encryption requires 2 modular exponentiations and the proof of validity, in an election wit c candidates, requires $4c$ exponentiations. The total cost for the voter, in bit operations, is $2 + 4c \log^3 n \approx \mathcal{O}(c \log^3 n)$. Each authority a involved needs to broadcast a proof of its private share, check the proofs of validity of the users and cooperate between them to decrypt the final tally. The total cost can be regarded as: $a + ac4 \log^3 n \approx \mathcal{O}(ac \log^3 n)$.

The encryption of the vote in the system presented by Juang et al. [2002], requires 1 modular exponentiation. Later, each elector needs to send its encrypted vote to each administrator and construct the signature from the responses she got. This requires 6 modular exponentiations. To vote she sends the vote through an untraceable e-mail system (mix-net) requiring m more exponentiations. The final cost is $(7 + m) \log^3 n \approx \mathcal{O}(m \log^3 n)$. To carry out the blinding signature scheme each authority performs 4 modular exponentiations, an special authority called the counter, performs 2 for each vote. Previous to this, an expensive preparation setup is carried out, we do not count it since it can be performed offline. The cost can be expressed as $4a \log^3 n + 2 \log^3 n \approx \mathcal{O}(a \log^3 n)$ bit operations. Baudron et al. [2001] define a hierarchical arrangement of authorities, the original paper uses three levels and we denote the number of levels as l. The elector needs to encrypt his vote for an authority in each level, requiring l modular exponentiations. She also needs to provide zero-knowledge proofs for every encryption, and an extra proof to show all the encryptions contain he same vote. This results in $l + 1$ zero-knowledge proofs, each requiring 6 modular exponentiations. Assuming l is a small constant, the cost can be computed as $(l + 6(l + 1)) \log^3 n = (7l + 1) \log^3 n \approx \mathcal{O}(\log^3 n)$. Authorities need to interact with the voter for the zero-knowledge proof, this requires 3 modular exponentiations. Each authority needs to compute its local result and forward it to its immediate superior authority. Authorities a in the same level must cooperate to partially decrypt the results, they also must provide a partial proof of decryption. The whole system cost can be expressed as $3a + aP_p \log^3 n \approx \mathcal{O}(a \log^3 n)$, being P_p the cost of the partial proof.

In Cramer et al. [1996], the elector needs to encrypt her vote, using 2 modular exponentiations and the proof of validity, which requires 4 exponentiations. The vote is split in t shares, the shares are sent to t different authorities together with a commitment. Each commitment requires 2 modular exponentiations. The cost for the voter is $(2+4+2t) \log^3 n \approx \mathcal{O}(t \log^3 n)$.

Each authority checks the proofs of validity of the users, this requires of 4 modular exponentiations. Each tallying authority checks the t shares posted by an authority employing a_2t modular exponentiations. The final cost can be computed as $(4a_1 + a_2t$, being $a_1) \log^3 n$ the authorities and a_2 the tallying authorities. Since t depends on the number votes, and $a1, a_2$ can be grouped in a authorities the cost can be expressed as $\mathcal{O}(a \log^3 n)$. In the system presented by Li et al. [2009a], the elector must interact multiple times with the different authorities to get verified, get a blind signature and cast her encrypted vote. This results in a total of 15 modular exponentiations to cast a vote: $15 \log^3 n \approx \mathcal{O}(\log^3 n)$. Authorities must generate a pair of public keys for each possible elector, but since this can be pre-computed we will ignore it in the cost function. Authorities must also generate another pair of keys for each registered voter, we will count the cost of finding the inverse in the RSA[2] which is $\mathcal{O}(\log^2 n)$ as we saw when analyzing TAVS. Furthermore the authorities nedd to perform 14 modular exponentiations to decrypt, verify, sign and re-encrypt each ballot. The cost of the system can be expressed as $\mathcal{O}(\log^2 n) + 14 \log^3 n \approx \mathcal{O}(\log^3 n)$.

Each voter needs to encrypt and sign her vote using ElGamal encryption in the system developed by Philip et al. [2011]. In addition to this, they also need to craft a zero knowledge proof for the vote. These processes require 5 modular exponentiations, the total cost for the voter remains constant $5 \log^3 n \approx \mathcal{O}(\log^3 n)$. Each authority a needs to provide a commitment of his share of the secret key, this uses 1 modular exponentiation. Each re-encrypted vote needs to be decrypted, requiring 2 modular exponentiations. The zero-knowledge proofs must be checked by each authority. Finally, the authorities cooperate to decrypt the valid votes, this process demands 2 modular exponentiations per vote. The total cost can be expressed as $(a+2+a+2) \log^3 n) \approx \mathcal{O}(a \log^3 n))$. Essex et al. [2012] report an exhaustive analysis of their system. We ignore the elevated number of modular exponentiations in the registration phase since it can be performed before the elections. To cast a vote and submitting a ballot and its credential, the elector needs to perform $(8c + 4) \log^3 n \approx \mathcal{O}(c \log^3 n)$ bit operations. The number of ballots b can be larger than the number of electors since they can generate fake ballots. The most expensive part of the system is the ballot authorization

[2]There are also other costs when computing RSA keys such as primality tests and the Euler's totient function, however we prefer to keep it simple for the comparison. We refer to the original work for more information about the voting protocol.

process. The total cost, in number of bit operations, can be expressed as $((4c+4)b+(280a+12ca+19a+2)b+3ac)\log^3 n) \approx \mathcal{O}(ac\log^3 n)$. Being a the number of authorities and c the number of candidates.

In Thi and Dang [2013], the elector has to communicate multiple times with the authorities to get identified and get her dynamic ballot. These communications are encrypted using public key cryptography. This results in 10 modular exponentiations for the user, with a cost of $10\log^3 n \approx \mathcal{O}(\log^3 n)$ bit operations. The cost of the system is defined by the checks and signatures of certificates (5 modular exponentiations) and the generation of keys for each registered elector. The total cost, in number of bit operations can be expressed as $5\log^3 n + 2\log^2 n \approx \mathcal{O}(\log^3 n)$. The elector assigns P points between c candidates in the ranked vote system introduced by Yang et al. [2018]. The vote is constructed as a matrix with c rows and $\log P$ columns. The elector must encrypt all the cells in the matrix, requiring 3 modular exponentiations each encryption. She also needs to provide a partial-knowledge proof for each cell, and an extra zero-knowledge ṫproof for the whole ballot. Partial-knowledge proofs require 8 modular exponentiations each and the zero-knowledge proof requires 5 modular exponentiations. Finally, the ballot must be signed, requiring 2 modular exponentiations. The work done by the elector can be summed up as $(8c(\log P)+5+2)\log^3 n \approx \mathcal{O}(c(\log P)\log^3 n)$. The proofs posted by the voters must be checked. The cost for partial-knowledge proofs is 6 modular exponentiations, and 4 for zero-knowledge proofs. In addition, all authorities must broadcast a commitment of their private key share, this requires 1 modular exponentiation. The performance of the whole system can be expressed as $(6c(\log P)+4+a)\log^3 n \approx \mathcal{O}(ac(\log P)\log^3 n)$. In Aziz [2019], the elector needs to request a token, check the signature of the ballot and cast her vote. This results in 5 modular exponentiations with a cost of $5\log^3 n \approx \mathcal{O}(\log^3 n)$ bit operations. The authorities must generate a token and generate a public/private key pair for each elector, and perform the blind signature. The total cost of the system can be expressed, in terms of bit operations, as $2\log^2 n + 14\log^3 n \approx \mathcal{O}(\log^3 n)$.

In Table 4.2, the methods presented by Li et al., Thao et al., and Aziz have the same theoretical time complexity as TAVS. Nonetheless, there are important details to take into account because directly affect to the performance of the methods (as we mentioned on Section 4.2.3, modular exponentiations dominate the time-complexity functions and hide other costs of the systems such as key generation, random permutations, other modular operation etc.)

Li et al. propose a method that requires a heavy public key infrastructure, generating an asymmetric key for each eligible elector and a second one for each registered elector at runtime. The use of four authorities increases the cost of processing a vote by a factor of seven when compared to TAVS. Although their signature step is similar to our proposal, our approach eliminates the need for the elector to depend on an authority to craft the ballot.

Aziz's method needs to generate tokens and keys during the election process, making the cost of processing a vote seven times higher than TAVS. Additionally, Aziz shows that voters must cast their votes through an anonymous channel, such as a mix-net, but does not measure the time complexity of this process.

Thao et al. also require key and dynamic ballot generation at runtime for each elector and rely on PET tests to verify elector identifiers, needing up to five authorities to operate the system. These factors almost triple the cost of processing a vote compared to TAVS.

In summary, TAVS does not need to compute keys per user and reduces the number of authorities required, making it the most efficient voting protocol among those presented. Furthermore, unlike other methods, the elector can independently create their own ballot and and only needs to interact with 2 authorities along the voting process.

4.3 Distributed Trust, a Blockchain Election Scheme

Historically, electronic voting systems have employed a public bulletin board to display the election process and the results. One of many examples is TAVS, as we covered in Section 4.2. They seek a way of publishing the voting information to later allow to verify and audit the election. In the last decade, blockchain technology has proved itself as an effective, distributed and decentralized public ledger Belotti et al. [2019], Paulavicius et al. [2019]. Thus, many electronic voting schemes have started to employ blockchain as a way to either have a decentralized, public and reliable bulletin board, or as a way to develop elections inside the blockchain itself as we discussed in Section 4.1.

Despite the efforts of e-voting to increase participation, such as remote and, in some cases, multi-platform access, most people do not trust electronic voting. It is difficult to trust in a system based on abstract mathematical results that people do not fully understand. E-voting should not be all about technological issues, but also about trust and about actively engaging all the involved parts in the democratic process. For this reason, we present an e-voting protocol in which traditional political parties, to whom people trust to some extend, take a dynamic and accountable side. We introduce a new voting scheme inspired by Monero's one-time public keys ($OTPK$s), and ring signatures that employs the blockchain as a bulletin board through which parties can publicly communicate. In our approach, each party is given partial power and full accountability in reaching a common interest despite their antagonistic nature. We develop a new voting scheme focused on trust, which is scalable, secure and preserves the anonymity and privacy of the electors.

4.3.1 Description of our Proposal

In this section, we present our voting scheme, called Distributed Trust. Our scheme presents a fully auditable, public, and distributed voting system that is based on blockchain technology. All information related to the voting process is registered into the blockchain, with each vote being a transaction. All blocks on the blockchain are open for public consultation. To foster greater engagement from electors in the voting process, we have assigned a unique role to political parties in our system. Political parties act as miners who listen and process transactions, and as such, any party wishing to participate must provide computing power. Despite their role in the voting process, electors need not place any trust in these parties, as the entire voting process is both public and auditable, ensuring privacy and anonymity. In many cases, individuals feel more confident when they have a party to hold accountable in the event of any issues arising. Our system takes this into account and ensures that there are appropriate mechanisms in place to address any concerns.

We employ a Proof of Authority (PoA) consensus algorithm Xiao et al. [2020], Barinov et al. [2018], as an alternative to Proof of Work. This variation reduces the computational cost required to run a functional blockchain. PoA is a permissioned blockchain where only some certified participants are allowed to carry out certain actions. In our system, parties are responsible for listening to transactions, verify the signature of the encrypted vote and make sure the vote is correctly written on the blockchain. Parties are the only partners with write access to the blockchain, while the rest of participants have read-only access. PoA can be seen as a variation of Proof of Stake Xiao et al. [2020], but instead of staking monetary tokens, parties stake their identity. The trust is distributed among reduced set of parties with adversarial interests, only these parties have write access to the blockchain. A Nash equilibrium is reached in where different entities collaborate because there is no reward in following a different strategy. Indeed, since our approach is fully logged and auditable and the parties are linked with real life entities, the penalization for indecorous conduct is immense. Their reputation will be discredited if they misbehave. Parties have write access, but since transactions are encrypted and anonymously signed, the user's privacy is not affected. By using PoA, our approach is faster since it does not require costly computation, environmentally friendly since it does not consume so much electricity and simpler to scale.

We consider the process as five different stages: election setup, registration, vote casting, vote processing and tallying. The system is discussed in detail in the next sections.

Election Setup

Before the election starts, parties must collaborate to generate the parameters of the election. To encrypt the votes electors send to the parties, we employ RSA, therefore, parties must generate the RSA parameters: the public modulo n, the public *verification key v and the private* signature key s. Since giving the private key s to a single party would result in giving up too much power, l parties apply the threshold RSA key generation protocol proposed by Damgård and Koprowski [2001]. This protocol relies on the work of Frankel et al. [1998] to remove the necessity of a trusted dealer. It introduces a (k, l)-threshold RSA sharing scheme in which the parameters are computed in a distributed way and the secret key is generated in l fragments. To recover the secret key, any subset of at least k parties can collaborate to find the original secret s. Even if some parties are corrupt, they would need to collaborate with at least k parties. The same applies to honest parties, k are required to recover the private key s.

We proceed to give an overview of the aforementioned key generation protocol. In RSA, modulo n is computed as the product of two large primes p and q. To distribute the trust, p and q are computed as the sum of different shares chosen by the parties: $n = (p_1 + p_2 + \ldots + p_i + \ldots + p_l)(q_1 + q_2 + \ldots + q_i + \ldots + q_l)$. To avoid from parties maliciously changing its share, they

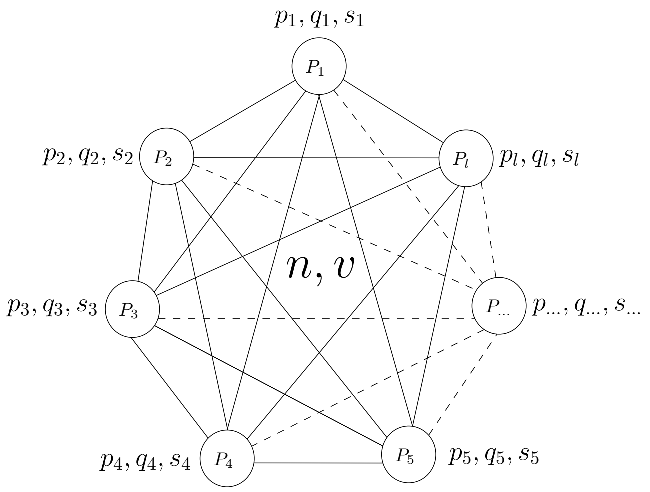


Figure 4.5: Distributed generation of RSA parameters. Generation of n and v is made of parties' additive shares p_i and q_i. Each party ends with a share of the private key s_i.

are required to publish a commitment (see Section 2.3.3) during the process. The distributed computation of n is carried out using a variant of Shamir's secret sharing (see Section 2.2). This variant from Frankel et al. [1997], also employs random polynomials in which the independent term is the secret, but it allows the computation of n to work outside prime fields.

Once the modulus is agreed, parties can jointly derive the public exponent v in a similar way, a distributed test division is required to test the primality of the candidates. When computing s, to prevent from revealing the shares of the polynomial, the shares v_i are used as an exponent to compute: g^{v_i} , being g a generator of the group defined by v. This way each party gets an additive share of the private key s_i. Then, they proceed to construct a (k, l)-threshold polynomial of s. Thus, a distributed generation of RSA, in which parties only have partial information of the private key s is achieved. To decrypt or sign a message, parties must collaborate to retrieve the shared private key. The collaboration process can be seen as a graph (see Figure 4.5), in which each node represents a party, and each edge represents the interchange of messages. The retrieving protocol is detailed in the following sections.

We require all the traffic of messages as well as the commitments related to any vote to be published as transactions on the blockchain. Thus, the blockchain comprehends all the information related to the election. RSA parameters n and v are also released to the blockchain. Besides from the distributed key, each party has its own personal pair of public/private keys. These keys are used to sign all the transactions they make. All this parameters of the election are published in the blockchain as the first block.

Registration

Electors must register before the election starts. For this purpose, a local administration defines the census of potential electors. It will also store and manage elector's keys as well as generate $OTPK$s for each elector. The administration will declare the process through which the electors will identify and register their public keys, and the end-time of the registration phase.

Any elector willing to participate will generate a two pairs of elliptic curve keys (a, A),(b, B). Then, the elector will follow the process specified by the administration, and, in case of correctly identifying herself, the administration will link their public keys (A, B) to their identity.

Once the elector registration term ends, the administration computes a $OTPK$ per elector using their public keys and a different random number r per $OTPK$. Next, the administration sends the information to the parties. Parties send a transaction through the blockchain, making public a list with the $OTPK$s of each public key pair and the associated R, such that $R = rG$. The owners of a given $OTPK$ either will be able to recover the corresponding private key computing $H_s(aR) + b$, or can be notified by the administration in order to save computation time. Where H_s is a hash function that maps to integers within the elliptic curve field.

The same transaction also includes general configuration information for the election such as: presented candidates or agreed encoding of the vote. This second transaction contains all the remaining public parameters of the election. Figure 4.6 illustrates the process electors carry out in order to register themselves.

Vote Casting

Once the electors have decided what to vote for, they must follow three steps to cast a valid vote. First they must encrypt it to protect its direction until the election is over. To do so, they read the public key v, and the modulo n from the blockchain and apply a modular exponentiation to encrypt it using

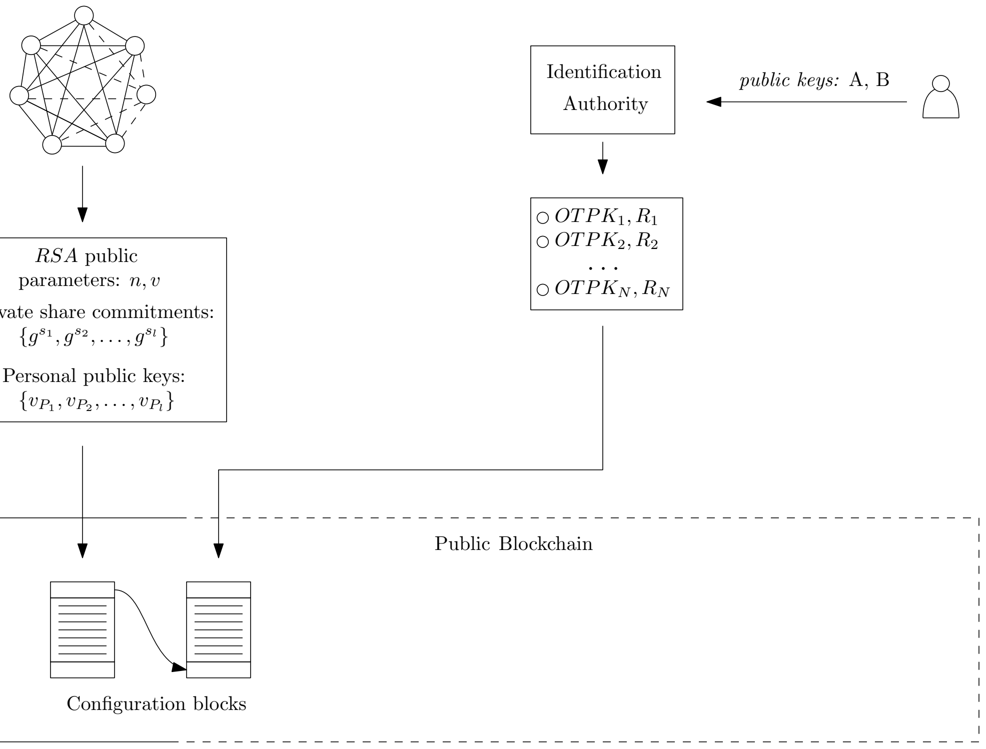


Figure 4.6: Electors register their public keys in a local identification authority. Then, the identification authority computes all the $OTPK$s. All the public parameters of the election are added as the first blocks of the blockchain.

RSA. Before encrypting the vote, the elector must select a fixed length random *mask* of her choice to concatenate to the vote. Otherwise, all the votes in the same direction will result in the same encryption. The elector will obtain an encrypted value such as $evote = (vote \| mask)^v \ mod \ n$.

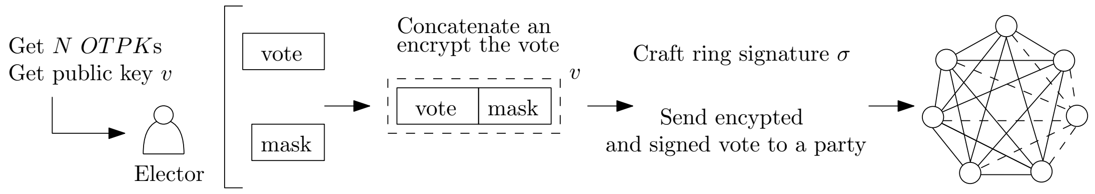

Get N $OTPK$s
Get public key v
Elector
vote
mask
Concatenate an encrypt the vote
vote
mask
v
Craft ring signature σ
Send encypted and signed vote to a party

Figure 4.7: Casting a ballot: Electors obtain election parameters from the blockchain. Then, they select a fixed length mask and concatenate it to the vote. Adjacent boxes represent concatenation, while the dashed box represents encryption using modular exponentiation. Electors craft a ring signature using the consulted $OTPK$s. Once the ballot is signed and encrypted, they encode it as a transaction and send it to any party of their choice.

Next, the elector must sign the vote to prove that it has been casted by an eligible elector. To perform the ring signature generation procedure described in Algorithm 6, the electors randomly take N $OTPK$s from the list of public keys, including their own. The elector obtains a ballot conformed by the encrypted vote and its signature: $ballot = \{evote, \sigma(evote) = (P, K, c_0, r)\}$. Note that the number of public keys in the ring N is tightly connected with the ambiguity of the signer. N acts as a security parameter, larger values imply more ambiguity, thus more privacy. However, it makes the signature slower and more expensive in terms of computation. A fixed or a minimum value for the ring size can be established depending of the security level and available hardware.

Finally, when the vote has been encrypted and signed, the elector sends the ballot through a blockchain transaction to a party of her choice or a random one. Figure 4.7 illustrates the casting process.

Vote Processing

Parties act like miners, listening to the blockchain network and expecting transactions addressed to them. When a party finds a transaction addressed to himself in the pool of unprocessed transactions, proceeds to verify the integrity of the ring signature. Figure 4.8 shows the processing of a vote.

When it has received enough transactions to fill the block size, the party creates a block that is added to the blockchain and later broadcasted to the rest of parties:

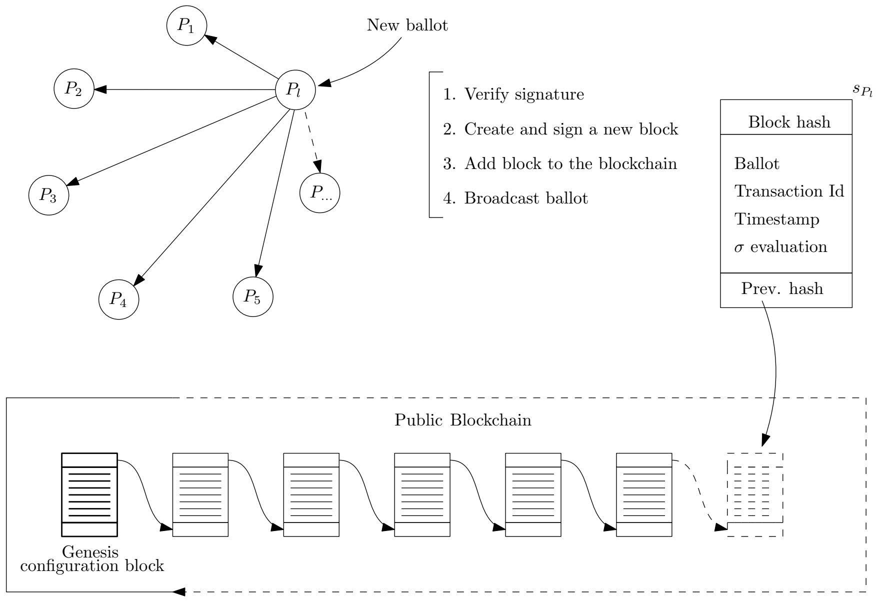


Figure 4.8: Processing of a vote. Each party is responsible of verifying received ballots, and ensuring they are correctly added to the blockchain. All the published blocks are by the party's personal private key.

1. It applies Algorithm 7 to the signature to verify its correctness.
2. It creates and signs a block with the following attributes:
    - Block Id.
    - Votes and transactions Id.
    - Timestamp.
    - Result of verifying the ring signature.
3. It adds the block to the blockchain.
4. It broadcast the new block to the rest of parties so they can also verify the votes.

For a more detailed and technical explanation about how the blocks are created and processed within the blockchain, we refer the reader to the Appendix C.

Tallying

Once the voting phase finishes, parties stop accepting transactions from electors and proceed to compute the final tally of the election.

First, parties must recover the secret key s to decrypt votes. To do so, a subset Λ of at least k honest parties collaborate to compute the original polynomial containing s. To recover a polynomial from k shares, defined as $s_0 = (x_0, y_0), s_1 = (x_1, y_1), \dots s_k = (x_k, y_k)$ we employ Lagrange interpolation, as it was introduced in Section 2.1.3. We know recover the equations that enable this interpolation:

$$\mathcal{L}(x) = \sum_{j=0}^{k} y_j l_j(x) \tag{4.2}$$

$$l_j(x) = \prod_{\substack{0 \le m \le k \\ m \in \Lambda \\ m \ne j}} \frac{x - x_m}{x_j - x_m} \tag{4.3}$$

Once we recover the polynomial $\mathcal{L}(x)$, s can be obtained as:

$$s = \frac{L(0)\Delta}{\Delta^3} \tag{4.4}$$

being Δ a factor used when generating the RSA key. Please note that $\Delta = l!$. It is used for computing the random polynomials when using Frankel et al. [1997].

When s is recovered, parties can decrypt the received votes and compute the final tally. Parties must analyze the received votes directly from electors and those received from another parties. This way, each party can compute a global and independent tally. If multiple votes contain the same key image, only the last vote will be considered valid.

When the tally is completed, each party has to publish a new block containing each received transaction, the result of checking the ring signature and the direction of the vote. At the end of the block they must add the results of the election and the private key of the election s. This message has to be signed by the parties. The private key is also published to allow electors to compute their own tally. If all the parties agree with the tally the election is finished. Figure 4.9 depicts the tallying phase.

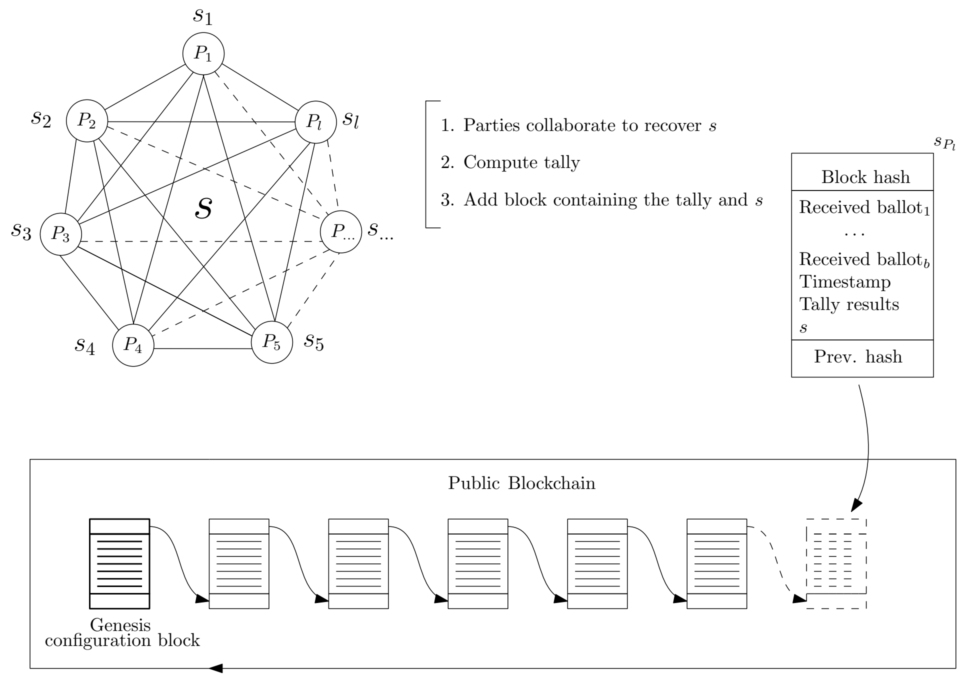


Figure 4.9: Parties collaborate to recover the private key s. Then, each of them individually computes a personal tally and adds it to the blockchain. It contains all the information needed by a third party to audit the tally.

4.3.2 Properties of the voting scheme

A voting scheme can be described by its properties, as defined in Section 4. These properties define what it can provide and under which circumstances. We devote this Section to study the security of Distributed Trust, and how it fulfills all these properties. Let us note our proposal is based on well-known cryptographic primitives, therefore most of the proofs rely on the underlying problems of those primitives.

Verifiability

Lemma 4.3.1. *Distributed Trust is end-to-end verifiable.*

Proof. Key images (see Section 3.3.2) work as a private receipt. Thus, allowing the elector to read from the blockchain to check her ballot was casted, recorded and tallied properly. As mentioned above, key images are anonymous and do not compromise elector's privacy.

Concerning universal verifiability, we note that any person, participant or not in the election process, is able to ensure that every vote has been tallied-as-recorded. This is achieved thanks to the public nature of the blockchain. Note that this does not ensure that the vote has neither been casted-as-intended nor recorded-as-casted because that would violate the privacy property.

Summarizing, our proposal provides universal verifiability by posting in the blockchain the key to decrypt the orientation of every vote recorded. Anyone can take the key and the votes stored in the blockchain to compute a tally by themselves. □

Accuracy

Lemma 4.3.2. *As RSA encryption system remains secure, the system is auditable by third parties, and, ring signatures are unforgeable, then Distributed Trust is accurate.*

Proof. We divide the proof in three parts:

(a) *No one can change anyone else's vote:* Votes are encrypted using RSA: assuming that no elector shares his secret key; that at last k parties are honest; and, that the Discrete Logarithm Problem (DLP) has no efficient solution for carefully selected parameters, the votes remain unaltered until the final tally.

(b) *All valid votes are included in the final tally:* Parties are responsible for processing and tallying all received valid votes. Given universal verifiability proved on Lemma 4.3.1, not including all valid votes would result in an early finalization of the election.

(c) *No invalid vote is included in the tally:* For a vote to be included in the final tally, its ring signature must be valid. Assuming that the

Elliptic-Curve-DLP(ECDLP) has no efficient solution, no signature can be forged.

□

Democracy

Lemma 4.3.3. *If the ECDLP is semantically secure, no one can impersonate a valid elector or perform double voting in the Distributed Trust voting scheme.*

Proof. The proof can be separated in two parts:

(a) *Eligibility:* Only electors in the census are able to register their public keys in the registration administration. The list of public keys and elector's identifier are stored in the blockchain to prevent the administration from creating fake electors. If the ECDLP problem is computationally secure. Then, only the registered elector can recover their personal $OTPK$ from the public list.

(b) *Double voting:* Key images work as a commitment of the private and public key of a elector. As stated previously, if the ECDLP has no efficient solution, then no modification of the key image can be made. Therefore, each elector is authorized, without revealing her real identity, to vote only once.

□

Privacy

Lemma 4.3.4. *If the ECDLP has no efficient solution, elector's identity remains private in the Distributed Trust electronic voting scheme.*

Proof. Key images, ring signatures and $OTPK$s are based on ECDLP. These are the only cryptographic constructs related to elector's identity. Assuming that, under the right parameters, no efficient solution exists for ECDLP, we can conclude there is no method to expose the elector's identity. Therefore, the elector is protected by the size of the ring signature, because any member of the ring is a possible signer with the same probability. We also stress that even if two equal votes were encrypted using the same ring of public keys, their signature will differ and privacy will be granted. □

Robustness

As mentioned above, in a (k, l)-threshold RSA key sharing scheme l represents the total number of involved parties, and k represents the minimum of collaborating parties needed to recover the secret key.

Lemma 4.3.5. *In a (k, l)-threshold RSA key sharing scheme, if at least $l - k + 1$ parties are honest, Distributed Trust voting scheme is robust.*

Proof. Parties are the only ones with write access, thus, electors cannot directly interfere in any of the processes stages. To recover the private key and compute the final tally at least k parties must cooperate. If a party (or any subset of them lower than k) misbehave, their actions are publicly auditable through the blockchain. Therefore they can be sanctioned and left out of the process without compromising the running election nor the final tally. Let us note, that even in the worst case of k malicious parties colliding to recover the secret key before the election ends, elector's privacy and the integrity of the vote will prevail since they depend on the ring signature bounded to each vote. They will be only capable of knowing the directions of the votes before the tally phase. □

4.3.3 Time complexity analysis

We devote this section to analyze the asymptotic computational time complexity of our election scheme, both for the elector and the involved parties and, second, to provide a comparison with the most similar systems reviewed in Section 4.1.

We consider here transactions per second (TPS) as a way to measure the throughput of our blockchain-based voting scheme. TPS are heavily influenced by the consensus algorithm employed: block proposal, block validation, and the mechanism used to solve forks among others. Here, we note that, by using PoA instead of Proof-of-Work, the limiting factor is no longer the block proposal but the block validation. Because no extensive hashing is required to propose blocks. Therefore, ring signatures, which are the main factor in block validation, determine the number of TPS. For a real implementation and empiric study of the time-complexity of ring signatures, please see Appendix B.

Next, we summarize the main stages of a consensus protocol and determine the associated computational time complexity. For computing the asymptotic time complexity, we consider the modular exponentiation, employed in RSA and the crafting/verification of a ring signature, as basic units. This operation is the most expensive and the basic construction units of our scheme. Modular exponentiation has a time complexity of $\mathcal{O}(\log^3 n)$ bit operations Menezes et al. [1996], being n the input and $\log n$ its size in bits.

Block Proposal

Thanks to PoA, no intensive computation is required to propose a new block, no resource-consuming hashing is employed with respect to Proof-of-Work. Every identified party who receives enough transactions to craft a block may propose a new one. Only a modular exponentation is required to sign the proposed block. Then, the complexity to propose a block can be expressed as $\mathcal{O}(\log^3 n)$.

Block Validation

The validation of a block requires to check the ring signature of every transaction contained within it. The computational complexity of this validation varies depending on the elliptic curve used and the size of the ring signature employed. Also note that we require all parties to validate all the new blocks, therefore the validation process is also affected by the number of involved parties. To validate the block, the signature of the party who created the block must also be checked. This results in one additional modular exponentiation. The asymptotic complexity of block validation can be expressed as $\mathcal{O}(tr_vp + \log^3 n)$, where t represents the number of transactions, r_v the cost of verifying a ring signature and p the number of parties involved in the consensus.

Information propagation

Message communication is another crucial factor to determine the throughput of a system. However, since we work with a permissioned blockchain, the number of nodes that send messages is minimal. A simple broadcast between parties is enough to communicate new blocks. Therefore, the number of messages remains linear with respect to the number of blocks b and the number of parties. Each message must be signed by the sending party, so the complexity can be regarded as: $\mathcal{O}(bp \log^3 n)$.

Block finalization

For a block to be finally accepted by all parties, it must be added in the longest chain. If two parties try to add a block at the very same time, a collision occurs. As we mentioned on the previous section, we follow the longest chain rule and each party is responsible for checking their blocks are correctly added to the blockchain. As the validation of a block does not depend on the previous block's hash, adding an already validated block for a second time does not require further computation. However, it is safe to assume that probably the block would be outdated and a new block proposal and its corresponding propagation should be carried out. The complexity can be expressed as $\mathcal{O}(qbp\log^3 n)$, being q the number of collisions requiring a new block re-added to the blockchain. The number of collisions is difficult to estimate since it depends on many empirical factors and varies from election to election. The worst case would be an election on which all the electors vote at the same time and the ballots sent are equally distributed between all parties. That would result in many concurrent collisions. Because of the unequal distribution of a real election, we can assume q will be low when compared with successful finalized blocks. Nonetheless, if the number of collisions becomes a problem, we could abandon following longest chain in favor of a byzantine fault tolerance consensus or a round-based writing Xiao et al. [2020].

Reward mechanism

In our adaptation of PoA, the reward and the purpose of the election scheme are the same thing. Parties are, allegedly, interested in carrying out the election process. Parties are motivated in an adequate election since their public reputations are at stake. Since the reward is abstract it does not affect the computational time complexity.

Tallying

For computing the final tally, parties must collaborate to recover the secret key and then proceed to decrypt all the votes. The collaboration requires p signed messages and the decryption needs one modular exponentiation per transaction(vote). The complexity of the tally can be expresses as $\mathcal{O}(pt\log^3 n)$.

Therefore, the total time complexity of the system can be expressed as $\mathcal{O}(\log^3 n) + \mathcal{O}(tr_v p) + \mathcal{O}(\log^3 n) + \mathcal{O}(bp \log^3 n) + \mathcal{O}(qbp \log^3 n) + \mathcal{O}(pt \log^3 n)$. Given that the number of blocks b depends linearly on the number of transactions t and that t dominates b, we can substitute b for t. After grouping some terms, the final complexity can be simplified as $\mathcal{O}(tp(r_v + q \log^3 n))$. Thus, the system's time complexity depends on the cost of verifying ring signatures and on the cost of running modular exponentiations, which are both affected by the number of involved parties, the number of collisions and the total transactions processed by the system.

We did not consider the cost of distributely generating the keys for RSA encryption because it can be done offline before the election process and it is carried out off the blockchain. Apart from the initial genesis blocks, which would have only required a pair of modular exponentiations, this process does not affect the complexity of our scheme.

Besides, one elector willing to craft and cast a vote, only needs to perform the encryption of the vote and the associated ring signature. The time complexity for the final user can be expressed as $\mathcal{O}(\log^3 n + r_c)$, being r_c the cost associated to craft a ring signature.

Comparative evaluation of systems

We devote this section to compare the performance of our proposal with the most similar ones studied in Section 4.1. Ring signatures and modular exponentiations determine the time complexity of our approach. For the rest of the comparison, and as we did for TAVS in Section 4.2, we employ modular exponentiation as the basic unit in the analysis.

We compare the asymptotic time complexity of the remaining works to prove the validity of our approach. When the methods do not specify a part of their protocol or do not provide enough information, we introduce a variable in the complexity analysis. Table 4.2 summarizes the associated complexity for the elector and for the protocol to process the received votes. For more details, we refer the reader to the original works. Protocols employing different kinds of ring signatures are compared, to provide a fair example we assume the time to craft r_c or to verify r_v a signature are comparable and can be agglutinated under the same variable despite their different implementations.

Note that the number of votes v and the number of transactions t are semantically equivalent, they both represent the number of processed votes.

	Elector's Cost	System's Cost
Salazar et al. [2010]	$\mathcal{O}(r(\log^3 n + r_c))$	$\mathcal{O}(vr(\log^3 n + r_v))$
Chen et al. [2008]	$\mathcal{O}(\log^3 n + r_c)$	$\mathcal{O}(vp(\log^3 n + r_v))$
Yang et al. [2020]	$\mathcal{O}(cs \log^3 n)$	$\mathcal{O}(cts \log^3 n)$
Distributed Trust	$\mathcal{O}(\log^3 n + r_c)$	$\mathcal{O}(tp(q \log^3 n + r_v))$

Table 4.2: Table representing the asymptotic time-complexity of the work performed by electors and the system in number of bit operations. In the table: r refers to the number of rounds in the case of round voting, v represents the number of votes, c refers to the number of candidates, s references the number of possible scores for each candidate in the case of ranked elections, t represents the number of transactions in a blockchain environment, and p the number of parties involved.

On the other hand, the number of candidates c and the number of parties p, not always represent the same partners. Not all protocols directly involve the candidates and some systems include extra authorities to handle credentials or distribute responsibility.

In Table 4.2, the results obtained by our proposal compete with the reviewed systems. Indeed, Chen's system and our proposal require the minimum effort for the final elector. Regarding system's complexity; it can be observed that, as for many works, it scales linearly with the number of involved parties and total number of votes. Salazar and Yang's works are also affected by other factors given that they support round and ranking e-voting respectively. Our proposed e-voting protocol is scalable due to its linear complexity and introduces the blockchain as a distributed public ledger without losing performance with respect to analogous works.

4.4 SUVS: Secure Unencrypted Voting Scheme

Traditionally, electonic voting schemes employ cryptography to encrypt and secure elector votes. But, as we covered in Section 2.5, there are alternatives to the well-known computational complexity approach. Our proposal, the Secure Unencrypted Voting Scheme (SUVS), is a voting scheme that maintains a high level of security, even in the absence of cryptography. Our approach is inspired by secret sharing schemes, as detailed in Section 2.2.

By fragmenting the vote into multiple pieces of information, we can conceal the original vote without revealing any sensitive information about it. Each individual piece of information alone is meaningless, and offers no clues as to the content of the vote. However, when the pieces are combined as a whole, the original vote can be reconstructed with confidence. Our approach therefore offers a secure and efficient method for conducting voting without relying on encryption.

Our objective is to design a new, user-friendly voting scheme that is both efficient and secure. We aim to engage sectors that may be averse to technology or resistant to change. By fragmenting the ballot into multiple pieces of information, no individual piece of data reveals any sensitive information about the vote. This unique approach ensures that the integrity of the vote is maintained while allowing for the decentralization of ballot processing responsibilities. Our proposal opens the door to the possibility of less conventional electronic voting schemes, with the goal of reducing the reluctance of novice users to engage in the electoral process.

4.4.1 Description of our Proposal

We devote this section to describe our *Secure Unencrypted Voting Scheme (SUVS)* scheme. SUVS requires minimum interaction from the elector and protects the vote using a constructive approach which does not require to encrypt/decrypt the votes.

The whole scheme is based on the generation of some pieces of information, which by separate do not reveal any information about the elector and her vote, but when combined, these pieces of information reveal the direction of the vote.

Our voting system comprises three distinct entities: electors who cast their individual votes; an identification authority (IA) responsible for verifying the eligibility of electors and certifying ballots in a blind manner; and political parties, who serve two purposes: firstly, to represent themselves as an option in the election, and secondly, to recover, validate, and tally the votes cast. To facilitate communication between the different entities, a Public Bulletin Board (PBB) is employed to provide public information about the election. As of today, numerous blockchain-based approaches (see Chapter 3) provide the necessary tools to create a public and distributed bulletin board.

SUVS consists on five sequential phases: the system setup; the ballot crafting; the ballot certification; the vote casting; and, the tallying phase. During these phases, the elector generates a private polynomial that serves as her own ballot. This polynomial allows the elector to conceal her vote as a set of points and to cast it during the corresponding phase. We take advantage of the properties of polynomial interpolation to ensure that the vote is impossible to recover unless all the points are known. In the final phase, political parties collaborate to recover the secret polynomial and its associated vote from the received points. Figure 4.10 illustrates the interactions between the different entities involved in the protocol, while Example 4.4.1 provides a step-by-step illustration of the processes leading up to the final tallying. In the following sections, we provide a detailed description of each of the five phases that make up our voting system.

System setup

Before the election process can begin, it is necessary to establish the methods that will be used to sign the electors' ballots. The IA is responsible for elector identification and ballot certification, which requires generating the parameters for setting up a digital signature scheme.

We employ blind signatures, as described in Section 2.6.1, to prevent double voting while protecting the privacy of the electors. This allows us to certify ballots from valid electors without linking their votes to their identities.

In the description of SUVS, we consider a RSA signature scheme to implement blind signatures because of its homomorphic properties under modular exponentiation, although any other method with similar properties could be used instead. To prepare for the blind signature scheme, the elector uses the public component of the IA signature key v and the public RSA modulus n, which will be used to certify the ballots.

The IA also states the hash function to be used, sets up the degree d of the polynomials to create, sets the maximum number of points the user can generate l, being $l > d$, as well as generates the prime p under which modulo operations will be carried out. Please note, that the degree of the polynomial d must be, at least, equal to $j - 1$, being j the number of involved parties on the election. This enforces the collaboration of all parties to recover the polynomial. Apart from this consideration, increasing d does not provide a greater level of security. We detail this issue later in Section 4.4.2.

When the setup phase ends, the IA publicly distributes v, n, d and p so that every elector can independently craft her own ballot.

Ballot crafting

Once the elector has decided her vote, she encodes it as an integer C. Then, she proceeds to generate a d-degree polynomial as shows Equation 4.5.

$$q(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_1 x + C \pmod{p} \tag{4.5}$$

such that the independent term contains the encoded vote C. Please note, that it is not necessary all the coefficients to be non-null to deal with a d-grade polynomial. Please also note, that coefficients must be smaller than the prime p, otherwise they will be reduced.

Next, the elector samples from the polynomial at least $d+1$ points, with a maximum of l points, see Equation 4.6. For the shake of clarity, we assume in the rest of the article the user generated the minimum of $d+1$ points. When required, we will order the set of points P taking into account the first coordinate, in order to transform, unequivocally, any set into a sequence of points. If we did not order the points, the hash function used to characterize them will provide non-deterministic hashes.

$$P = \{(x_1, q(x_1)), (x_2, q(x_2)), \ldots, (x_{d+1}, q(x_{d+1}))\}. \tag{4.6}$$

Anybody who knows P can interpolate the original polynomial $q(x)$, and therefore recover the vote. This set P is actually, the ballot itself. To cast it, it will be splitted in shares to be sent to the parties. In order to allow the reconstruction of the splitted vote, the elector digest the sorted set of points P using the hash function selected in the system set up phase. Please, once again note the importance of the points being sorted for the hash function to produce consistent outputs. For this reason, we define a function *shash*, that sorts, accordingly, the received input before applying the hash. The output of the *shash* functions acts as a *commitment* of the points. This commitment acts as a receipt that ensures the set P has not been tampered, and demonstrates the validity of the ballot when signed. An outline of the crafting process is depicted in Algorithm 11.

Algorithm 11 Ballot crafting

```
1: Elector : Codification of the vote as an integer $C$.
2: Elector : Generation of a private polynomial $q(x)$ such that $C$ conforms
   its independent term:
       $q(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_1 x + C$
3: Elector : Generation of $P$, an ordered sequence of $d+1$ points:
       $P = \langle (x_1, q(x_1)), (x_2, q(x_2)), \ldots, (x_{d+1}, q(x_{d+1})) \rangle$
4: Elector : Generation of a random mask. Crafting of the ballot:
       $b = hash(sort(P)) \cdot mask^v \bmod n.$
```

Ballot certification

In order to prevent double voting, each elector is required to send her ballot to the IA to be properly certified. To avoid the possibility of relating the ballot with the elector, our proposal blinds the ballots before sending them to the authority. To do so, the elector selects a random invertible *mask*, considers the verification key v published by the IA and computes the ballot b as shown in Equation 4.7.

$$b = shash(P) \cdot mask^v \bmod n, \tag{4.7}$$

which is sent to the IA, together with her identification, through a secure channel.

The IA checks if the identification is valid and the elector is on the census of registered voters. If the identification is correct, the IA proceeds to sign the ballot as referenced in Equation 4.8

$$\begin{aligned} b^s &\equiv (shash(P) \cdot mask^v)^s \pmod n \\ b^s &\equiv shash(P)^s \cdot (mask^v)^s \pmod n \\ b^s &\equiv shash(P)^s \cdot mask \pmod n \end{aligned} \tag{4.8}$$

Just to clarify, the mask is a randomly generated number that is used to blind the ballot before it is signed by the IA. The purpose of this is to prevent the IA from learning the contents of the ballot while still being able to certify it as valid. Once the masked ballot is signed, the IA cannot know the original contents of the ballot unless the mask is revealed to them. After the signature process, the IA replies the elector through a secure channel with the signed ballot. The IA also publishes on the PBB, a tuple of the form $\langle id, b = shash(P) \cdot mask^v)^s \rangle$. The goal of this is twofold: it allows the elector to check that her ballot was received as intended, and it proves that every ballot comes from a valid identity from the public census.

The elector receives the signed ballot and proceeds to recover the signed commitment which will certify her vote. Note that, the elector is the only one who knows the mask and its inverse. Hence, the elector obtains the signed commitment as indicated in Equation 4.9.

$$b^s \cdot mask^{-1} \equiv shash(P)^s \cdot mask \cdot mask^{-1} \equiv shash(P)^s \pmod{n} \tag{4.9}$$

Then, the elector obtains the certified signed commitment that will be used in the next steps of the election. These phases are detailed on Algorithm 12. Note that, despite requiring her identity in order to sign the ballot, the IA has no means to link the commitment with the elector. The elector is also able to check if the signed ballot was tampered during the process, as she is able to independently verify the integrity of the signed commitment.

Algorithm 12 Ballot certification

```
Require: $d_{elector} \leftarrow$ Elector's identification.
Require: $b \leftarrow$ Ballot to be certified.
1: $Elector \rightarrow IA : \langle id_{elector}, b \rangle$
2: if $id_{elector}$ is valid then
3:     $Elector \leftarrow IA : b^s \mod n$.
4: else
5:     $Elector \leftarrow IA : Error$.
6: end if
7: $Elector$ : Recover the signed hash:
       $$b^s \cdot mask^{-1} \equiv shash(P)^s \pmod n$$
8: return Certified hash: $shash(P)^s$
```

Vote casting

The elector has a set P that can be used to recover the direction of her vote, and the signed commitment of the set $hash(P)^s$ which identifies her vote as a valid one.

To finally cast the vote, the elector sends a partition of P (shares of the ballot) together with the certified commitment to all the parties implied in the election tallying. Note that a basic property of polynomial interpolation states the impossibility to recover a d degree polynomial with d or less points of such polynomial. This allows to send different shares of information to different parties with certainty that no information of the original vote will be revealed, unless all the parties collaborate to do so.

Therefore, taking into account that k parties are implied in the election, the elector partitions the set P into k non-overlapping subsets SP_i, such that P results as the ordered merge of every SP_i. See Equation 4.10.

$$P = \left\{ \bigcup_{\forall i \ 1 \leq i \leq k} SP_i \right\} \tag{4.10}$$

Each one of the parties receive a share SP_i of P together with the hash and the certified hash: $shash(P), shash(P)^s \pmod n$. Please note that the certified hash operates as digital signature of the hash, and that both are sent and needed to check its validity. Also note that the elector can freely decide which subset is sent to each party.

We also note, that it is possible to reduce the number of shares to put aside from the process those parties which receive no share of the elector's ballot. Note also that this does not affect to the validity of the vote but to the transparency of the process. However, we force that every party receives one share. By doing this, we ensure that all parties must collaborate to recover the vote. Thus, no subset of malicious parties will be able to recover the vote before the tallying phase.

Once all the subsets have been sent, the vote has been cast. Note that, unlike what happens in the ballot certification phase, no personal information goes along with the shares of the vote. Hence, parties have no means to associate the received shares with an elector's identity. Please note that, while parties cannot identify voters, they can reject invalid messages by leveraging the digital signature of the commitment. Invalid signatures must be rejected, and this can be used as defense against DDOS attacks. The casting process is depicted on Algorithm 13.

Algorithm 13 Casting a vote

```
Require: $shash(P)^s \leftarrow$ Certified commitment.
Require: $P \leftarrow$ Set of points.
1: $Elector$ : Create $k$ non-overlapping shares $SP_j$ of the ballot $P$ such that:
        $$P = \left\{ \bigcup_{\forall j \ 1 \le j \le k} SP_j \right\}$$
2: for each $party_j$ in the election do
3:     $Elector \rightarrow party_j : \langle shash(P), shash(P)^s,\ SP_j \rangle$
4: end for
5:
6: return One share $SP_j$ for each party $j$.
```

Tallying

Once the election phase is over and no new votes are accepted, the parties can proceed to reconstruct the votes and compute the final tally.

In first place, the parties consider the certification that accompanies the shares to find the set of shares, each one of them received by a different party, that allow to reconstruct each ballot P. Note that the original sequence can be easily obtained, by ordering the set P, and that is possible to check the validity of the certification.

A set of points P such that its certified commitment $shash(P)^s \pmod{n}$ is not correct, or that its cardinality exceeds the defined maximum, $|P| > l$, is discarded. Parties can now, individually, use the points in P to recover the original polynomial $q(x)$ which contains the vote as its independent term C. To recover a polynomial from a set of points:

$$P = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_j = (x_j, y_j)\}$$

we suggest to use Lagrange's interpolation, introduced in Section 2.1.3, and represented again in Equations 4.11 and 4.12.

$$q(x) = \sum_{i=0}^{j} y_i l_i(x) \tag{4.11}$$

where:

$$l_i(x) = \prod_{\substack{0 \leq k \leq j \\ k \neq i}} \frac{x - x_k}{x_i - x_k} \tag{4.12}$$

that, when $q(x)$ is interpolated, allow to recover the encoded vote C as illustrated in Equation 4.13.

$$C = q(0) \pmod{p} \tag{4.13}$$

Please note that the interpolation operations are not carried out modulo p. When we forced, on the ballot crafting, the coefficients of the polynomial a_d to be smaller than p, we made sure that we would interpolate the exact same polynomial without the need of modular arithmetic. Otherwise, we could interpolate a different polynomial, congruent mod p, but with a different encoded vote C.

The parties publish on the PBB, a 3-tuple per vote containing: the certification of the ballot (i.e. the signed commitment); the ballot itself (i.e. the set of points P that conceal the ballot); and the reconstructed vote C. The final tally obtained by each party can also be published. The PBB is

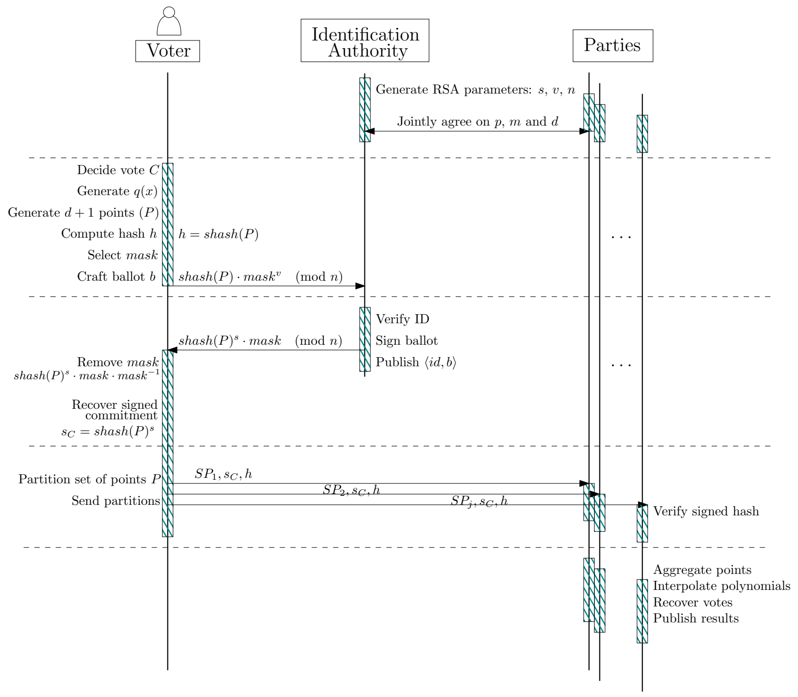


Figure 4.10: Scheme representing the election phases, its associated agents and their message interchange.

available to everyone to check that their votes have been counted as intended, and to verify the integrity of the final tally. Algorithm 14 shows the steps to compute the final tally.

Algorithm 14 Tallying votes

```
1: Parties collaborate to obtain all the shares of each ballot by merging and
   ordering points with the same $shash(P)^s$.
2: Verify the signed commitment $shash(P)^s$ corresponds to the ballot and
   check that the cardinality of $P$ does not exceed the predefined maximum
   number of points $l$.
3: Parties (independently) interpolate the original polynomial $q(x)$ using
   the $d+1$ points in the ballot.
4: The vote $C$ is recovered and published on a public bulletin alongside the
   received $shash(P)^s$ and the set of points $P$.
```

Example 4.4.1. *To illustrate the process, let us consider an election scenario with three candidates vying for the position. In order to set up the system, an AI generates an RSA signature key with the public key $\langle v, n\rangle$ and private key $\langle d\rangle$. The specific values of the key and the hash function h used, are not of particular significance in this example and will be referred to as is. The authority then announces that a degree of 4 and a modulus of $p = 47$ will be used in the process.*

After the system has been set up and the values have been published to the electors, each user encodes their vote as an integer modulus p in order to prepare the ballot. In this example, the vote is encoded as $C = 23$. The elector then generates a polynomial randomly with the codification of their vote as the independent term. Let us consider the following polynomial as an example:

$$q(x) = 11x^4 + x + 23 \bmod p,$$

that is now used to generate five random points. Let these points ordered with respect to the first coordinate be the following:

$$P = \langle (7, 27), (13, 12), (29, 45), (31, 17), (45, 9)\rangle.$$

This set of values is actually the ballot the elector will split into pieces. To obtain a commitment of the ballot, the elector computes $shash(P)$ and masks the digest using a private invertible integer modulus n:

$$commitment = shash(P) \cdot mask^v \bmod n,$$

the elector then sends the result, along with their identification, to the IA to obtain a blind signature. It is worth noting that there is no way for the authority to discern the direction of the vote since it only receives the masked commitment.

Once the usual checks are carried out, the authority computes the certificate signing the commitment using its signature private key:

$$pre\_certificate = (shash(P) \cdot mask^v)^s \bmod n.$$

As explained above, the elector can easily obtain the certificate of the ballot as:

$$certificate = pre\_certificate \cdot mask^{-1} \bmod n = shash(P)^s \bmod n.$$

The ballot is then split into three disjoint pieces (the number of candidates), for instance:

$$P_1 = \langle(29,45),(45,9)\rangle$$
$$P_2 = \langle(7,27)\rangle$$
$$P_3 = \langle(13,12),(31,17)\rangle$$

which are sent to the respective candidates together with the certificated commitment. It is worth noting that it is impossible for the candidates to discern the direction of the vote unless they all agree to share the points for the purpose of interpolating the polynomial. Furthermore, it is infeasible for any candidate to manipulate the ballot since they lack the necessary information to do so.

To reconstruct the ballot, candidates use the certified commitment to select the corresponding pieces of the same ballot, verify their integrity, and interpolate the polynomial using whichever available method.

4.4.2 Properties of the voting scheme

We now analyze the security properties, as they were defined in Section 4, of SUVS. We enumerate the desired properties and prove how our system meets them. Please note, that the security of our scheme, except for the blind signature in the registration phase, does fall under the information-theory paradigm. Therefore, and unlike other computational security-based schemes, we do not need require traditional security parameters. Once some basic constraints are achieved (e.g. $d \geq j - 1$), the overall security does not rely on larger values to become more secure.

Verifiability

Lemma 4.4.2. *SUVS voting protocol is end-to-end verifiable.*

Proof. In order to prove the lemma, we will prove first that after the identification stage, the elector can verify that her vote was not tampered during the certification of the commitment, and, second, that at the end of the voting process, that any vote was correctly recorded and tallied.

First, we note that during the process of commitment certification, the elector is the only one who knows the mask that conceals the commitment. Therefore, only she can remove the mask and apply the public key of the identification authority check if the commitment was correctly certified or it was somehow modified. She also can check on the PBB that the masked commitment received by the IA has not been tampered.

Second, regarding universal verifiability, any individual (the elector included) can consult on the PBB all the individual votes, that is: their set of points, and their certified commitment. This information is enough for any interested party to check the validity and integrity of the votes.

Thus, our method allows anyone to compute the tally, verifying the validity of all the individual votes, and, therefore, auditing the election process. □

Privacy

Lemma 4.4.3. *SUVS guarantees the elector's privacy.*

Proof. To prove the privacy property, we will prove first, that the identification authority cannot compute the direction of any elector's vote, and, second, that parties can neither do so.

First, note that the elector mask that conceals the commitment of the vote is a private election, the identification authority has not information to unmask the commitment, and, therefore, the authority cannot gain knowledge of the elector's commitment. Thus, the identification authority has no way to, once the PBB is made public, relate elector's identification to her final vote.

Second, we note that the parties do not receive any personal information from the elector. Hence, they can no relate the vote in any way. □

Democracy

In SUVS, the IA responsibility is twofold: it verifies elector's identification to check that they are eligible, and, prevents double voting by using blind signatures.

Please note, that even if the IA acted maliciously, democracy would still be ensured. The received ballots and its associated identity on the public census are published on the PBB for everyone to review. A malicious IA could not forge invalid votes or linked valid ones to their elector. However, in a different scenario, an attacker could try to bypass IA's certification and forge the employed blind signature scheme.

Lemma 4.4.4. *As the RSA scheme employed for blind signature remains secure, our electronic voting protocol is democratic.*

Proof. For a vote to be considered valid, it must be accompanied by a certified commitment. This certification is carried out by the IA through a blind signature scheme based on RSA. Assuming that the Discrete Logarithm Problem (DLP) has no efficient solution for meticulously selected parameters, the attacker has no means, apart from bruteforce, to break the signature scheme. □

Accuracy

Lemma 4.4.5. *SUVS voting protocol is accurate.*

Proof. We prove the lemma in three independent steps:

1. *Valid votes are counted:* Parties are responsible of processing all valid received votes. Both individual and universal verifiability (Lemma 4.4.2), guarantee that electors can check that all valid votes are included on the final tally.

2. *No invalid votes are considered:* In order a vote to be considered valid, it needs to be correctly certified. As proved on Lemma 4.4.4, the creation of fake votes is unfeasible. Thanks to verifiability, electors also can audit the results and check that all individual votes are valid.

3. *Vote modification:* Any modification to any of the unencrypted shares of a vote, will be detected because it will make the certified commitment not match the shares. As long as the hash function used remains secure, no tampering of the votes will succeed.

□

Robustness

We first prove that SUVS is robust with the only condition that one party remains honest. Then, we prove that it is unfeasible a coalition of users could tamper with new votes.

Lemma 4.4.6. *Robustness of SUVS is ensured if at least one contendant party remains honest.*

Proof. We note that parties must cooperate to interpolate the polynomials and recover the final votes. If at least one contendant party remains honest, the remaining parties have no way to interpolate the polynomial and access to the vote. Of course, at a cost of a high reputational loss, a set of parties can misbehave or refuse to cooperate, in which case the election would have no tally. □

Lemma 4.4.7. *It is unfeasible that, according SUVS protocol, a coalition of electors could tamper with new votes.*

Proof. We note that a coalition of malicious electors could take profit of the multiplicative properties of modular exponentiation, and use their certified commitments in order to obtain a new tampered one h_t. Nevertheless, as long as the chosen hash function remains secure, it is unfeasible to obtain a set of points P_t, such that $shash(P) = h_t$, and such that the polynomial interpolated using P_t codifies in its independent term a valid vote. □

Perfect secrecy

Perfect secrecy, as defined in Section 2.5.3, provides security by uncertainty. The concealing of the vote by partition SUVS is based on provides security derived entirely from information theory, creating a system where partial information does not reveal anything about the scheme's secrets.

As stated on Lemma 4.4.3, there is no mechanism to relate an elector to her vote. By providing perfect secrecy, we also ensure that, even in post-quantum scenarios, it is impossible to reveal any information on any vote unless all the parties are malicious or compromised.

Lemma 4.4.8. *SUVS provides perfect secrecy and its encoding is resistant to post-quantum attacks.*

Proof. If an attacker gains knowledge about all the shares of a vote but one, there is no way he could interpolate the polynomial generated by the user and, therefore, gain access to the direction of the vote.

Note also that, under modular arithmetic, there exists a combinatorial number of d-degree polynomials consistent with d points. This implies that, even if the attacker could find a polynomial that covers the points and encodes a valid vote, he could never be sure which one was the original vote encoded by the elector. □

Although this assertion holds for a computationally unbounded attacker, in a situation where the adversary has access to quantum queries and can execute superposition attacks, Shamir's secret sharing scheme may inadvertently divulge the secret Damgård et al. [2013]. Notably, this attack is feasible if, and only if, the attacker successfully corrupts a minimum of $t/2$ parties in a (t, n)-threshold secret sharing scheme. Otherwise, the scheme remains secure.

Malicious Elector Resistance

A malicious elector may try to craft a ballot in such a way that she can achieve double voting.

Lemma 4.4.9. *SUVS is resistant to malicious electors.*

Proof. SUVS defines the structure of the ballot as:

$$b = shash(P) \cdot mask^v \mod n$$

In order to achieve double voting, a malicious elector could generate two different set of points P_1, P_2 and craft the ballot as follows:

$$b = shash(P_1) \cdot shash(P_2) \cdot mask^v \mod n$$

To the IA, the resulting ballot looks the same as a valid one. So it signs the ballot and returns it to the malicious elector. Then, the malicious elector proceeds to remove the mask

$$shash(P_1)^s \cdot shash(P_2)^s \cdot mask \cdot mask^{-1} = shash(P_1)^s \cdot shash(P_2)^s \mod n$$

Nonetheless, the malicious user cannot separate $shash(P_1)^s$ and $shash(P_2)^s$ since s is not known by the users. The malicious user would not only not be able to double vote, she would lose the ability to cast a single vote. □

4.4.3 Time complexity analysis

This section is devoted to the analysis of the time-complexity of SUVS. We demonstrate that our protocol is highly efficient and requires minimal effort from the involved parties. We differentiate between the computational complexity related to each individual elector and the complexity of the entire system to process all votes. We have chosen bit operations as the unit for our time-complexity analysis. As usual, n denotes the input operands, and $\log n$ denotes the number of bits in the input.

In order to certify and cast a vote, the elector needs to carry out a series of steps, including generating a polynomial, sampling some points, computing a hash function, and selecting some subsets. However, we neglect some of these steps in the time-complexity analysis for two main reasons: firstly, most of these steps can be performed offline before the actual election process starts; and secondly, they are not relevant in terms of time-complexity analysis because other operations dominate the overall complexity. For instance, the sorting of a sequence or a hash of length j is not comparable to the magnitude of $\log n$, and hence, its complexity can be neglected when compared with the other operations.

We only consider the mask generation and multiplication and exponentiation operations. They are the most expensive operations and dominate the time-complexity costs.

Ballot crafting

The user must generate a mask and search for its inverse to certify the ballot. To find the inverse using the Euclid's algorithm which has a complexity of $\mathcal{O}(\log^2 n)$ bit operations. Only one iteration of the algorithm is needed to find an invertible mask, otherwise it would mean that we broke RSA, i.e. finding a factor of n.

To craft the ballot the user must perform a modular exponentiation (See Line 7 on Algorithm 12), which has a cost of $\mathcal{O}(\log^3 n)$ bit operations. Additionally, a modular multiplication ist also required on Section 4.4.1. This operation has cost of $\mathcal{O}(\log^2 n)$ bit operations.

Then, the complete time-complexity for an elector to cast a vote can be expressed as:

$$\mathcal{O}(\log^2 n) + \mathcal{O}(\log^2 n) + \mathcal{O}(\log^3 n) \approx \mathcal{O}(\log^3 n) \tag{4.14}$$

SUVS Complexity

Now, let us consider the complexity of the whole system. To process a vote, the SUVS must apply a blind signature to the ballot, and at the end of the election, it must interpolate a polynomial from a set of points. We do not consider the compilation of the shares with the same certified commitment in the complexity analysis since the shares can be indexed and compiled in constant time.

- Ballot certification requires one modular exponentiation, which has a cost of $\mathcal{O}(\log^3 n)$ bit operations.
- Interpolation using Lagrange's interpolation requires a linear number of non-modular operations. Addition and subtraction present a time-complexity of $\mathcal{O}(\log n)$, while multiplication and division have a complexity of $\mathcal{O}(\log^2 n)$ bit operations. These operations are affected by the number of shares j (equivalent to the number of parties), as can be observed in Subsection 4.4.1. The complexity of this step, can be expressed as $(j(\mathcal{O}(\log^2 n) + \mathcal{O}(\log n)))^2 = j^2(\mathcal{O}(\log^4 n) + \mathcal{O}(2\log^3 n) + \mathcal{O}(\log^2 n))$. However, since j is known to be a low and bounded number, we can treat it as a constant and simplify it in the complexity as $\mathcal{O}(\log^4 n) + \mathcal{O}(2\log^3 n) + \mathcal{O}(\log^2 n) \approx \mathcal{O}(\log^4 n)$

Thus, the total time-complexity of SUVS scales linearly with the number of processed votes v and can be expressed as:

$$v(\mathcal{O}(\log^4 n) + \mathcal{O}(\log^3 n)) = v\mathcal{O}(\log^4 n) \tag{4.15}$$

Please also note, that the complete number messages sent by the elector in SUVS can be expressed as $1 + j$, while the total number of messages interchanged between the involved parties is simply j. These figures shows that the message complexity of SUVS is not affected by the fact of not encrypting the votes. Additionally, any additional messages that may be required for the user are non-blocking communications. The elector sends the shares and does not require any additional communication, and no future steps are delayed by this aspect.

System Comparison

We also present a comparison between the most similar systems reviewed in Section 4.1. The purpose of this comparison is twofold. First, it allows the reader to directly assess the reviewed systems. Secondly, it provides a common baseline to compare the performance of SUVS.

The nature of SUVS makes difficult to compare it with previous ones reviewed in the literature (see Section 4.1). Thus, in this case, we focus the comparison on the feature that can affect most the practical behavior of the protocol, the average number of messages sent in the protocol. Messaging between electors and parties is accessible to measure and network delays can surpass computational times. The results of our analysis are depicted in Table 4.3.

	Elector's Messages	Total Messages	Cryptographic Primitives
Chaum [1982],	2	$2v$	Blind Signatures
Li et al. [2009b]	5	$3v$	Blind Signatures
Thi and Dang [2013]	4	$7v$	Blind Signatures
Larriba et al. [2020]	3	$2v + 1$	Blind Signatures
Cramer et al. [1997]	1	$v + j$	Homomorphic Prop.
Yang et al. [2017], Yang et al. [2018]	2	$2(v + j)$	Homomorphic Prop.
Tornos et al. [2014]	3	$3v + 1$	Ring Signatures
Chen et al. [2008]	2	$2 + v + 2j$	Ring Signatures
Yang et al. [2020]*	2	$2j$	Homomorphic Prop.
Gao et al. [2019]*	2	v	Ring Signatures
Larriba et al. [2021]*	2	$2 + v$	Ring Signatures

Table 4.3: The table represents the number of messages sent by the elector and the whole system. In the table: when the number of authorities is not fixed, they are represented as j in the table, v represents the number of processed votes, and the $*$ symbol represents systems that are deployed over blockchain technology.

4.5 Review of the 3 Voting Protocols

After presenting and analyzing our three voting protocols, we now devote this section to provide a high-level comparison between the three of them. Since each one has different assumptions, and uses distinct cryptographic primitives they can be used in different scenarios depending on the requirements.

TAVS It is based in an RSA blind signature scheme and requires 2 non-colliding and unrelated entities, which might not be feasible in all scenarios. As a way to tackle this situation, which might be an issue if these 2 antagonic entities do not exist, we can employ our implementation based on smart contracts (See Appendix A) which reduces the assumption to a single entity. One of the greatest benefits of TAVS is the simplicity of its architecture, specially when compared to other works in the literature with similar goals. This efficient architecture with only 2 authorities is very feasible to scale to larger elections. The overall clarity of TAVS is also beneficial when dealing with people reluctant with e-voting protocols due to their complexity.

Distributed Trust Introduces the traditional political contenders within the election process and leverages blockchain, both as a decentralized PBB and as a way to carry out the election itself. It requires the parties to run their own blockchain, with limited capabilities, which might expensive in some scenarios, but builds an honor game where the reputation of parties it is at stake. This factor also increases the usability and accessibility to more traditional factions. Ring signatures are also a great ambiguity mechanism that preserves privacy even if the IA became malicious. By using blockchain technology, voters are enabled to verify the election state in any device and develop in top of it to build indexers and event listeners for the election.

SUVS Also introduces political contenders into the election process. SUVS requires the elector to take an active role in the ballot crafting: sampling the shares and deciding how to sent them to the authorities. Thus, providing a higher degree of security. The cryptogaphic primitives used allow to obtain post quantum resilience and demonstrates the feasibility, and power, of systems based in alternative systems not backed by complexity theory.

4.6 Conclusions

We covered three electronic voting protocols. These schemes tackle the challenge of e-voting from different angles, employing distinct cryptographic primitives, and with varying assumptions, as we discussed in Section 4.5. But all of them accomplish the required properties that a voting protocol should satisfy that were covered in Section 4. By providing a detailed time-complexity analysis for each one of them, we proved that our voting proposals are lightweight and allow for an efficient scaling. This allows for accommodating elections af any size.

We also provided a formal definition of the structure of a blockchain, that can be used to carry out whole elections (such as our Distributed Trust protocol) or used as PBB, in Appendix C. Besides the theoretical analysis of our voting protocols, we also provided implementations and real-world performances of ring signatures and the TAVS protocol in Appendixes B and A respectively. These do not only reinforce and demonstrate our theoretic analysis, but also provide the interested reader with a Proof-of-Concept framework to tests and evaluate the protocols. All the implementations have been open-sourced to facilitate future research.

4.6.1 Future Work

As for future challenges and possible extensions for these protocols, it would be interesting to study alternative identification methods, such as the use of Merkle trees and zero-knowledge proofs, that would make unnecessary to include a centralized identification authority in the protocols.

For Distributed Trust and SUVS, a possible extension could be focused on reducing the number of interactions the elector needs to carry out with the parties without affecting the reliability and trustworthiness of the system. This way, if an elector does not trust a certain party, she can decide to arrange and send the required messages without considering that specific party. This would require to ensure the availability and decentralization of existing parties to preserve the security properties, while also respecting elector's preferences within some threshold parameters.

In the same vein as we did for TAVS, and in order to empirically test our other proposals in real scenarios, a complete Proof-Of-Concept for SUVS and Distributed Trust is considered a valuable future goal.

Chapter 5

Identification and Distributed Access

Jamás se descubriría nada si nos considerásemos satisfechos con las cosas descubiertas.

Séneca

AZ2hA6DLQ/wxotw41m5vosDt
PhRcPDSd1g1VyZW/mtQrVQg
jOmhD82S54c3l8dMTlnUcbnND
zE4j6Rf/zj7GK4P5i1eMdPMhA
C4yL5lxVIYnRPVgQ22xezY=

Chacha20, k = >\x88Å\x1eh\x9aõÑ6A\x07¼\x96 ©a7ÖPBíÈ\x0cÃ¶\x92\x03\x080 %È©!, nonce= Ec/finMEEZU=

Identification is the process by which a user, or an entity, (from now on refereed as the Claimant) provides relevant information to guarantee, beyond any reasonable doubt, to a third party (hereinafter refereed as the Verifier) they are who the claim to be. This process is generally related with some kind of authentication policy to get access to a guarded resource. Identification also plays a crucial role in access systemsFerraiolo et al. [1995], Sandhu and Samarati [1994], voting schemes, and, controlled reward systems among others. See Section 5.7 for an example of its direct applications. We devote this chapter to introduce the open problem of identification and to present our three protocol to address anonymous registration.

Among all the optics in which identification can be studied, general anonymous identification Dodis et al. [2004] still remains an open problem since it involves what at first looks like an oxymoron: determine an identity as valid while respecting its privacy. Nonetheless, identification is an abstract concept that might have different meanings in separate contexts. Therefore, many works in the literature have studied identification and its associated problems under different relaxations of anonymous identification.

5.1 State of the Art

Let us note that our work, and the other articles here reviewed, are not related with identity-based cryptography Shamir [1984], Boneh and Franklin [2001]. Identity-based cryptography, as proposed originally by Shamir, defines a new asymmetric cryptographic scheme in where the public key can be any random string which is intrinsically associated to the user's identity, and hence the name of the approach. Our work is neither related with key-anonymity protocols Bellare et al. [2001]. These protocols aim to provide anonymity of the key under which the encryption was performed, even under chosen-cipher texts attacks. However, the provisioned anonymity is only against a third-party observer, and does not include the sender. Lastly, the Claimant/Verifier scheme might remember to the Prover/Verifier scheme used in Zero-Knowledge proofs (see Section 2.4). However, while Zero-knowledge can be a crucial part in many identification schemes, proving that having some information is not the same as to proving the identity in many contexts.

The conventional method for user authentication involves a registration phase where the user provides their credentials to a centralized entity. After successful registration, the user is granted access to the desired service. However, this approach has its drawbacks as it centralizes all the user's data in a single point of failure and requires the user to trust the entity responsible for authentication and access key management. To address these issues and enhance user privacy, researchers have explored the possibility of distributing user authentication among multiple servers. This distributed approach offers improved resilience against server breaches and offline attacks, especially if the servers do not store hashed information. Additionally, by using a proper threshold scheme, it enhances overall availability, thereby mitigating the risk of service disruption. Many works have studied distributed token generation through public key threshold signatures Desmedt and Frankel [1989], Santis et al. [1994], Gennaro et al. [2008] and threshold authentication codes Boneh et al. [2013], Naor et al. [1999] to protect the master key against server security breaches. Also, a different line of work has studied the use of threshold password authenticated secret sharing Camenisch et al. [2014], and threshold password-authenticated key-exchange MacKenzie et al. [2002], Raimondo and Gennaro [2003], Abdalla et al. [2005], as an improvement against the offline dictionary attacks present in more traditional password-authenticated key-exchange Katz et al. [2001], Bellare et al. [2000].

The methods mentioned before aim to enhance user privacy by distributing the authentication process, but they do not provide a complete decentralization of trust in the system. In contrast, our proposal offers three protocols that ensure certain authorities cannot link a user's identity to their actions within the system once they gain access. This added layer of privacy protection is a critical feature that prevents potential abuses of power or unauthorized data access by authorities. By implementing our protocols, users can trust that their privacy is protected, and authorities are held accountable for their actions within the system.

To tackle the trust distribution problem, Agrawal et al. present a password-based threshold authentication protocol (PASTA) in Agrawal et al. [2018]. PASTA is a general framework for token generation which distributes the task among a set of servers, such that any subset of t, can verify and generate tokens, while no subset of $t-1$ servers can forge invalid tokens. In their work, Agrawal et al. aim to shield the token generation process against server breaches as well as to reduce the number of interactions needed with respect to more traditional password sharing works. After a one time registration, the user provides the credentials to the token-generation servers. If the credentials are valid, then the servers respond with a part of the token. Otherwise, the protocol stops. Once the user has interacted with all the servers, the user proceeds to construct the token with the received parts. To prevent from offline attacks, the authors employ a threshold oblivious pseudorandom function (TOPRF) Freedman et al. [2005], Jarecki et al. [2016] on the server side. Agrawal et al. propose, analyze, and implement PASTA as a complete and general framework compatible with multiple TOPRF functions and various threshold token generations algorithms (both for symmetric and asymmetric cryptography). To the best of our knowledge, this is the most similar work to the protocols described in this chapter, but, while PASTA does not pay attention to the anonymity of users, we are mainly focused on their anonymity.

Anonymous Credential Systems (ACS) Chaum [1985] were proposed to provide a scheme where users can obtain access-keys directly from organizations, without the need of third-parties. In an ACS framework, organizations only know users by pseudonyms and users may have various unlinkable pseudonyms. Through digital signatures and zero-knowledge proofs, ACSs provide a set of functionalities such as: unforgeability; anonymous validation of credentials; unlinkability; or access-key transference between organizations while preserving user's privacy. There exist several works that propose original approaches with different features Chaum and Evertse [1986], Chen [1995], Damgård [1988], Lysyanskaya et al. [1999]. Nonetheless, as a consequence of the heavy use of zero-knowledge proofs, ACS are usually complex in terms of cryptographic primitives and have high time complexity. These factors are an inconvenient for ACS.

Among the papers in the literature we distinguish the one by Camenisch and Lysyanskaya, Camenisch and Lysyanskaya [2001], where they propose an ACS with interesting properties such as non-transferable access-keys, optional revocable anonymity, and one-show access-keys. Despite their focus on making a practical system, their extended protocol still needs up three rounds of interaction and heavy use of modular exponentiation operations. Despite these drawbacks, the proposed system was later implemented in Camenisch and Herreweghen [2002]. Also, in Belenkiy et al. [2009], Belenkiy et al. propose a delegatable ACS, that presents a hierarchical system in which access-keys can be structured in levels as an intent to model real word interactions.

Identity Management Systems (IdMs) have been developed to integrate the latest advancements in biometric and secure hardware technology with identification systems. These systems aim to provide a comprehensive suite of tools that handle all aspects of the identification ecosystem, including integration with biometrics, mobiles, and hardware identification devices. In order to maintain user privacy, these systems often rely on approximations such as ACSs. Due to the complexity and importance of their function, IdMs are typically backed by government or official institutions. In Moreno et al. [2019], Moreno et al. present an IdMs based on PASTA that provides unlinkability through distributed identity providers and biometric identification. In Bernabé et al. [2020], Bernabe et al. evaluate ARIES, an European IdMs that also includes ID-proofing based on biometrics and breeder documents handling within their framework. We do not mention IdMs as a comparable result, but as the current framework that includes identification methods similar to ours. We note that the scope of our work is much more limited than a complete IdMs.

5.2 Anonymous Access

A less restrictive scenario than anonymous identification, but also powerful, aims to provide *anonymous access* to members of a collective. The paradigm is shifted from granting access without knowing the identity of the user, to granting access to a set of identified users without knowing which user is accessing. This modification mitigates the problem of verifying an anonymous digital identity, and if the collective is large enough the user can obtain a high degree of anonymity. To provide an unambiguous definition of the anonymous access process, we differentiate the following three steps:

- *Identification* as the initial process by which a verifier certifies that the credentials presented by a claimant are valid. And therefore the claimant is entitled to access the resource or the service.
- *Registration* as the intermediate step in which the claimant obtains some access-key (e.g: access token, password) that he will later use to anonymously access the resource.
- *Access* as the final step where the claimant provides the access-key to a verifier to get access to the resource.

This differentiation allows to clearly detach the identification phase, where the claimant reveals the information that identifies him as an individual, from the registration phases and the proper access to the resource. Anonymous access aims to provide secure and mathematically sound methods that remove any traceability between these phases. Thus, the access-keys do not reveal any information about the claimants. To further ensure the anonymity of the process, decentralized protocols are preferred since they remove a single point of failure over a centralized entity.

To highlight the benefits from its decentralized nature let us describe a problem instance where a set of guards safeguard a resource. The access to the resource is granted if, and only if, all the guards certify the presented anonymous access-key is valid. Guards do not trust each other and only trust the verification process of the access-key.

We present three different non-interactive (see Section 2.4.2) protocols based on Shamir's secret sharing (see Section 2.2) that provide anonymous registration and one-time access under different scenarios. One-time access prevents the guard's need from having any kind of saved state on the claimant information, without losing the generality of the service. The requirements to access the guarded resource more than once can be checked in the identification stage, while leaving the access step completely anonymous. This stateless approach, together with the non-interactive property that minimizes messaging round-tripping, and, the lightweight computation the methods are based, make our presented protocols able to work in embedded environments with limited resources.

- Our first protocol, introduced in Section 5.3.1, presents the most restricted scheme where a trusted centralized entity is responsible for the identification and dealing with the access-keys.

- The second protocol, introduced in Section 5.4, improves the first approach and distributes the responsibility between a set of distributed authorized parties.

- Our third protocol, introduced in Section 5.4.2, further enhances previous approaches and provides a fully decentralized and anonymous registration and access.

The first two protocols present unconditional security and quantum resistance properties, since their security is based on information theory and not on complexity theory (See Section 2.5). Therefore, not even a quantum attacker with computationally unbounded capabilities could break their security. It is important to highlight, as mentioned in Section 4.4.2, that a quantum attacker who also has access to quantum queries and can execute superposition attacks may potentially compromise the underlying Shamir's secret sharing scheme if they manage to corrupt $t/2$ parties.

Our third method preserves the privacy of the claimants also in post-quantum scenarios and remains secure, meaning that no access-keys can be forged, as long as the discrete logarithm problem remains unsolved.

5.3 Centralized Registration, Anonymous Access

No matter which identification scheme is proposed, there must be some instant of time when the users provide their identification in order to prove the right to access a protected resource. In this section, we propose a first protocol that considers a central authority (assumed honest), a set of guards commissioned to control the access; and a set of accredited users[1]. The central authority has a dual role in this context. Firstly, it is responsible for establishing the framework by distributing access keys to users and assigning distributed access control to guards. Secondly, it serves as an audit authority to address any identification issues that may arise between users and guards. It should be noted that the creation of the required private channels is not within the scope of this paper. Interested readers may refer to the literature (e.g. Rescorla [2018]) for further details on how to implement them.

In the following, we will use the notation G to refer to the set of N guards responsible for controlling access to the resource, while C will denote the group of users seeking to access it. It should be noted that subsets of both users and guards may exhibit malicious behavior. The Centralized Registration Anonymous Access scheme can be defined as a system comprising five probabilistic polynomial time algorithms Koblitz and Menezes [2015]. These algorithms are outlined below:

- $SysSetup(1^k, N, C) \rightarrow (\{q_i\}_{i \in G}, pp)$. This algorithm generates a key-generation system $q(x)$ and some public parameters pp. The key-generation system is then randomly partitioned into $\{q_i\}_{i \in G}$ shares, which are privately distributed among the guards. Only the Central Authority has access to whole system $q(x)$. The set of shares of $q(x)$ meet the satisfiability condition described below.

- $Registration(id) \rightarrow a_{key}$. This algorithm generates for a correctly identified user a private access-key a_{key}.

- $AccessRequest(a_{key}) \rightarrow \{i, k_i\}_{i \in G}$. This algorithm is a call to the set of guards, who generate partial keys k_i according their share of the key-generation system. All the guards are committed to share their results with the rest of guards in G.

[1]The credentials owned by the users are supposed to be delivered beforehand by some trustworthy institution.

- $Combine(\{i, k_i\}_{i \in G}) \rightarrow tk$. The algorithm considers the partial keys computed by the guards and obtains an access-token by combining them.

- $GrantAccess(a_{key}, tk) \rightarrow 1/0$. Checks it the token tk corresponds to access-key a_{key}, in which case the guards agree to grant access (output 1).

Consistency is guaranteed if for any valid id and $a_k \leftarrow Registration(id)$, it is hold that:

$$GrantAccess(a_k, Combine(AccessRequest(a_k))) = 1.$$

It is assumed that *SysSetup* and *Registration* are exclusive algorithms of the Central Authority, which is trustworthy in this protocol.

It is important to highlight that the system ensures two important aspects. Firstly, only the central authority possesses the ability to link a user's identity with the access keys they have been issued. Secondly, the system does not provide any information regarding the usage of the resource by users, thereby preventing individual actions from being tracked. However, it is important to note that if malicious guards and the central authority were to collude, they may be able to combine their information and gain knowledge of user actions. Nevertheless, we assume that the central authority is honest in this context.

5.3.1 Trusted Registration, Anonymous Access

Next, we detail our protocol for trusted registration and anonymous access (TRA2). The setup of the protocol implies the generation of a random prime p, and a m-degree polynomial modulus p $(m < p - 1)$. Let us note that we are interested in the maximum degree of the polynomial, therefore, in order our proposal to work it is not required to consider a polynomial with all the coefficients non-null:

$$q(x) = a_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x + a_0 \bmod p.$$

Provided $q(x)$, the authority partitions the polynomial into N polynomials:

$$\begin{aligned}
&q_1(x) = a_{1,m}x^m + a_{1,m-1}x^{m-1} + \ldots + a_{1,0} \bmod p\\
&q_2(x) = a_{2,m}x^m + a_{2,m-1}x^{m-1} + \ldots + a_{2,0} \bmod p\\
&\vdots\\
&q_N(x) = a_{N,m}x^m + a_{N,m-1}x^{m-1} + \ldots + a_{N,0} \bmod p
\end{aligned}$$

such that the partitioned polynomials complement each other in order to obtain the coefficients of $q(x)$, that is:

$$a_i = \sum_{1\leq j\leq N} a_{j,i}, \quad \forall i,\ 0 \leq i \leq m.$$

The partitioned polynomials are communicated and assigned to the respective guards in a secure manner. It should be noted that unless all the guards agree to collude, it is impossible for any subset of guards to interpolate the polynomial $q(x)$.

Once the system has been established, users can request their access keys. Users provide their identifications to the authority, who utilizes the polynomial to generate a random point on $q(x)$, which serves as an anonymous access-keys for the user to access the resource while remaining anonymous to the guards.t́

It should be emphasized that unless the number of registered users reaches the bound determined by m, it is impossible for any group of users to successfully interpolate $q(x)$ and compromise the system by tampering with new access keys. Furthermore, the size of the modulus p has not influence in the security of the protocol, but it must be greater than m. Additionally, that the use of modular arithmetic bounds the size of the access-keys while does not prevent the possibility of dealing with a reasonable high number of users[2].

Once the access-keys have been delivered, in order for a user to access to the resource or service, he sends his access-key (point of $q(x)$) to all the guards (as illustrates Figure 5.1). The guards must collaborate in order to decide whether the access-key is correct or not. Thus, given any user u and his point $\langle x_u, q(x_u)\rangle$, access should be granted whenever:

$$q(x_u) = \sum_{1\leq i\leq N} q_i(x_u) \bmod p$$

[2]A value of m of 40 bits is not a big issue in terms of time complexity and is far enough to provide access-keys to all the inhabitants in Earth.

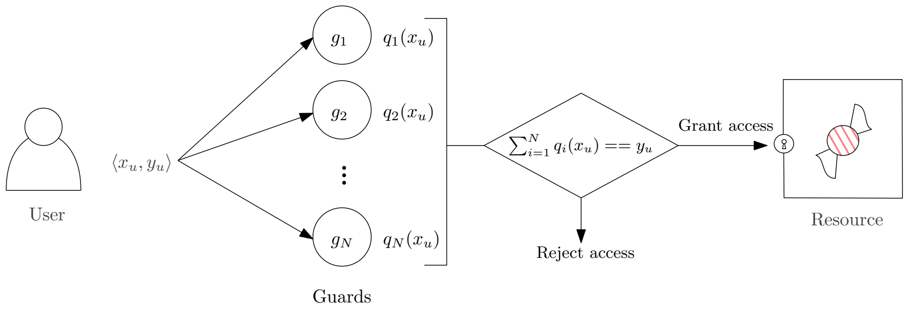


Figure 5.1: Users present their access token to the guards to gain access to a resource. The guards then collaboratively determine the validity of the provided token in a distributed manner.

In order to prevent the guards misusing the checked access-keys (for instance by distributing them to unauthorized users), we consider access-keys are single-use. Thus guards should also check the uniqueness of x_u to verify the validity of the request. The central authority is responsible for not generating two points with the same x coordinate. An outline of the complete protocol for our *trusted registration, anonymous access* protocol (TRA2) is summarized in Algorithm 15 and illustrated in Example 5.3.1.

Example 5.3.1. *Consider a scenario in which there are three guards responsible for controlling access to a particular resource, denoted by $N = 3$. Let us define $p = 7919$ and $m = 665$ as the public integers used to establish the system. Furthermore, let us also define:*

$$q(x) = 45x^{665} + 22x^{13} + 54x^7 + 1 \bmod 7919$$

, as the polynomial the authority (privately) generates in order to generate the access tokens. The final step to set up is to partition the polynomial $q(x)$ into three complementary polynomials to distribute among the guards, for instance:

$$q_1(x) = 26x^{665} + 4x^7 + 1 \bmod 7919$$
$$q_2(x) = 22x^{13} \bmod 7919$$
$$q_3(x) = 19x^{665} + 50x^7 \bmod 7919$$

Given that $q(21) = 655$, a valid access token for a correctly identified user could be of the form: $\langle x_u = 21, y_u = 655 \rangle$. Once the guards receive the access

Algorithm 15 TRA2 Algorithm. Trusted registration, anonymous access protocol.

```
1: System setup
2:     (a) Central authority sets a prime $p$ and generates a $m$-degree polynomial $q(x)$, where $m$ is greater than the maximum number of users to identify.
3:     (b) Central authority partitions $q(x)$ into $N$ polynomials $P = \{q_1(x), q_2(x), \ldots, q_N(x)\}$ that, for any $x$, meet the condition:
           $$q(x) = \sum_{1 \le i \le N} q_i(x) \bmod p$$
4:     (c) Central authority distributes a polynomial in $P$ to each of the $N$ guards.
5: User (trusted) identification and registration
6:     (a) Users send their identification to the central authority.
7:     (b) Central authority verifies the identification credentials.
8:     (c) If valid, central authority generates a random $x_u$ and replies to the user with $\langle x_u, y_u = q(x_u) \bmod p \rangle$.
9: Anonymous access
10:    (a) User sends her access-keys $\langle x_u, y_u \rangle$ to each one of the guards.
11:    (b) Each guard computes $q_i(x_u)$.
12:    (c) Guards check if $y_u = \sum_{1 \le i \le N} q_i(x_u) \bmod p$ .
13:    (d) If the access-keys are valid, access is granted.
```

token, each one can work out the result from its polynomial share:

$$q_1(21) = 26x^{665} + 4x^7 + 1 \bmod 7919 = 7526$$
$$q_2(21) = 22x^{13} \bmod 7919 = 3501$$
$$q_3(21) = 19x^{665} + 50x^7 \bmod 7919 = 5466$$

and must collaborate in order to check whether the token is valid or not. Indeed, $y_u = q_1(x_u) + q_2(x_u) + q_3(x_u) \bmod p$*, and access should be granted.*

5.4 Distributed Registration, Anonymous Access

The TRA2 protocol offers an attractive solution for anonymous access to a resource, particularly in situations where an honest authority is easily identifiable. owever, it is worth noting that the reliance on a central authority, assumed to be honest, could limit its applicability in certain scenarios. To overcome this limitation, we propose a modification to the protocol that eliminates the need for a central authority altogether. By distributing both the access and registration processes, our proposed protocol offers a more decentralized solution to the problem..

We define the Distributed Registration Anonymous Access scheme as the following system of seven probabilistic polynomial time algorithms:

- $SysSetup(1^k, N, C) \rightarrow pp$. This algorithm generates the public parameters pp for the scheme. These can be, either the result of an agreement, or randomly generated.

- $DealersSetup(pp) \rightarrow \{q^j\}_{j \in G}$. The algorithm is run by every dealer d_i that, privately, generates an access-key generator $q_{d_i}(x)$. This generator is randomly partitioned into shares, which are privately distributed among the guards. The set of shares of $q(x)$ meet the satisfiability condition described below.

- $GuardsSetup(\{q_i\}_{i \in D}) \rightarrow q^{g_j}(x)$. The algorithm is run by every guard g^j that considers the shares received from the dealers to obtain its own share of the key-access generator $q^{g_j}(x)$.

- $Registration(id, u_x) \rightarrow a_{key}$. The users call this algorithm to ask for a set of partial keys from the set of dealers. If correctly identified, the user receives a set of partial access-keys, the combination of which returns the user's access-key a_{key}.

- $AccessRequest(a_{key}) \rightarrow \{i, k_i\}_{i \in G}$. This algorithm is a call to the set of guards, who generate partial keys k_i according their shares. All the guards are committed to share their results with the rest of guards in G.

- $Combine(\{i, k_i\}_{i \in G}) \rightarrow tk$. The algorithm considers the partial keys computed by the guards and obtains an access-token by combining them.

- $GrantAccess(a_{key}, tk) \rightarrow 1/0$. Checks it the token tk corresponds to access-key a_{key}, in which case the guards agree to grant access (output 1).

Consistency is guaranteed if, for all k, for any valid id and $a_k \leftarrow Registration(id)$, it is hold that:

$$GrantAccess(a_k, Combine(AccessRequest(a_k))) = 1.$$

5.4.1 Trusted distributed registration, anonymous access

In this section, we present a protocol that distributes both the registration and the issuing of access tokens, as well as the access to a resource. To achieve this, we adopt the same principle utilized in the TRA2 scheme to distribute the access control. This enables us to replace the central authority with a set of registration authorities, referred to as dealers, denoted by D. In this protocol, termed *trusted distributed registration, anonymous access* (TDRA2), dealers are assumed to be honest.

Access control is still performed by N guards, who operate in the same way as in the TRA2 protocol.

To setup the system, the dealers first agree the modulus p, as well as the m-degree of a polynomial $q(x)$. Once the main parameters are chosen, each dealer d_i independently generates a m-degree polynomial $q_{d_i}(x)$. In the same way the access-keys of TRA2 protocol were points of a m-degree polynomial, we will consider the polynomial $q(x)$ which results of the sum of all the $q_{d_i}(x)$ polynomials generated by the dealers. Note that $q(x)$ is unknown to each individual dealer.

To ensure that the polynomial $q(x)$ remains unknown to the guards, each dealer d_i partitions their polynomial $q_{d_i}(x)$ into N complementary polynomials $q_{d_i}^{g_j}(x)$, such that:

$$q_{d_i}(x) = \sum_{j=1}^{N} q_{d_i}^{g_j}(x),$$

then, dealers proceed to securely send one of their shares to each guard. Each guard g_j considers the shares from all the dealers and computes its polynomial from the received shares as:

$$q^{g_j}(x) = \sum_{i=1}^{D} q_{d_i}^{g_j}(x),$$

and from this moment on, guards are ready to accept access access-keys. Figure 5.2 depicts a simple example of the process.

Note that, taking into account the shares of the polynomial known by the guards and the dealers, the following condition is met:

$$\sum_{i=1}^{D} q_{d_i}(x) = \sum_{i=1}^{N} q^{g_i}(x)$$

that is, dealers and guards consider different shares of the same polynomial $q(x)$.

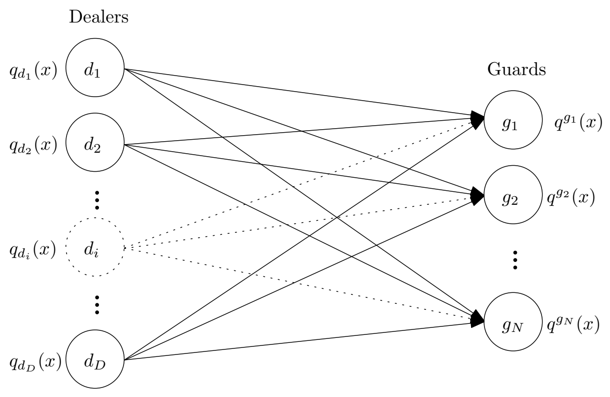


Figure 5.2: Dealers split their polynomial in N parts and distribute one piece to each guard. Guards use these pieces to construct its own private polynomial.

In order to obtain their access-keys, users first choose a unique value x_u and send their identification along with x_u to every dealer. Each dealer d_i checks the credentials to identify the user and verifies that x_u has not been used before. If the credentials are valid and x_u is unique, dealer d_i computes $q_{d_i}(x_u) \bmod p$ and replies to the user with the result.

$$\langle x_u, y_u = \sum_{i=1}^{D} q_{d_i}(x_u) \bmod p \rangle.$$

Figure 5.3, illustrates the process (TDRA2 and TRA2 access stages are the same). Algorithm 16 describes the protocol and Example 5.4.1 depicts it.

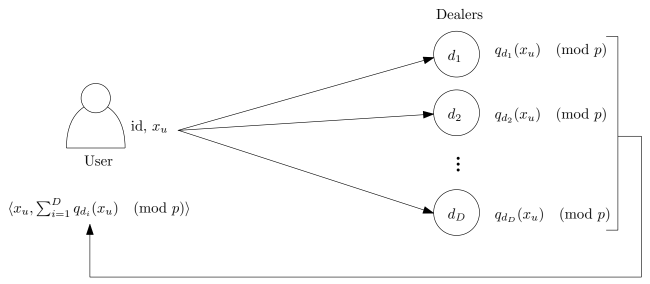


Figure 5.3: User sends her identification details, and selected x_u to the dealers. Dealers respond with the result of applying its polynomial if the identification is valid. Then, the user is capable of independently construct his access token from the response.

Example 5.4.1. *Let us consider an scenario with two dealers ($D = 2$) and three guards to control the access to some resource ($N = 3$). We consider the same public integers considered in Example 5.3.1 ($p = 7919$ and $m = 665$). Let also consider the following polynomials (privately) generated by the dealers in order to generate the access tokens:*

$$q_{d_1}(x) = 26x^{665} + 54x^7 + 1 \bmod 7919$$
$$q_{d_2}(x) = 19x^{665} + 22x^{13} \bmod 7919$$

The final set up step implies each dealer to partition his polynomial $q(x)$ into three complementary polynomials to distribute among the guards. For instance, consider the partition of the first dealer polynomial as:

$$q_{d_1}^{g^1}(x) = 6x^{665} + 1 \bmod 7919$$

$$q_{d_1}^{g^2}(x) = 10x^{665} + 54x^7 \bmod 7919$$

$$q_{d_1}^{g^3}(x) = 10x^{665} \bmod 7919$$

Algorithm 16 TDRA2 Algorithm. Trusted distributed registration, anonymous access.

```
1: System setup
2:     (a) Dealers jointly agree on a prime $p$ and $m$.
3:     (b) Each dealer generates an $m$ degree polynomial $q_{d_i}(x)$ modulus $p$.
4:     (c) Each dealer partitions its polynomial into $N$ shares and sends each
       one to a different guard.
5:     (d) Every guard $g_j$ sums the received components to compute its ($m$-
       degree) polynomial $q^{g_j}(x)$ modulus $p$.
6: User identification
7:     (a) Users send his identification along with a selected $x_u$ to each dealer.

8:     (b) Each dealer verifies the identification credentials and check that $x_u$
       is unique and has not been used before.
       If so, each dealer $d$ replies to the user with $\{x_u, q_d(x_u) \mod p\}$.
9:     (d) Users can compute their credential using the received points from
       the dealers as:
           $\langle x_u, y_u = \sum_{i=1}^{D} q_{d_i}(x_u) \bmod p \rangle.$
10: Anonymous access
11:     (a) User sends his access-keys $\langle x_u, y_u \rangle$ to each one of the guards.
12:     (b) Each guard computes $q^{g_j}(x_u)$.
13:     (c) Guards check if $y_u = \sum_{1 \le j \le N} q^{g_j}(x_u) \bmod p$.
14:     (d) Access is granted if the access-keys are valid.
```

, and the partition of the second dealer as:

$$q_{d_2}^{g^1}(x) = 19x^{665} \bmod 7919$$

$$q_{d_2}^{g^2}(x) = 10x^{13} \bmod 7919$$

$$q_{d_2}^{g^3}(x) = 12x^{13} \bmod 7919$$

which implies that the polynomials the guards consider are:

$$\begin{aligned} q^{g_1}(x) &= 25x^{665} + 1 \bmod 7919 \\ q^{g_2}(x) &= 10x^{665} + 10^{13} + 54^{7} \bmod 7919 \\ q^{g_3}(x) &= 10x^{665} + 12^{13} \bmod 7919 \end{aligned}$$

A user who sends her identification together with $x_u = 233$ *to the dealers will be replied with the values* 5048 *and* 6449 *from dealers one and two respectively, and will be able to obtain his access token as:*

$$\langle x_u = 233,\ y_u = 5048 + 6449 \bmod 7919 = 3578\rangle.$$

Once the guards receive the access token, each one obtains the following results from their polynomial shares:

$$q^{g_1}(233) = 4372; \quad q^{g_2}(233) = 4794; \quad q^{g_3}(233) = 2331$$

and must collaborate in order to check whether the token is valid or not. Actually, $q^{g_1}(x_u) + q^{g_2}(x_u) + q^{g_3}(x_u) = 3578 = y_u \bmod p$*, and the token is correct.*

This approach avoids the polynomial $q(x)$ to be stored in a single point of failure, and therefore, may lead users to increase their trust in the system. In Section 5.5.2 we prove the security of the protocol.

5.4.2 Anonymous registration, anonymous access

It is worth mentioning that although TDRA2 protocol effectively distributes the responsibility of controlling access, and prevents the possibility of the dealers or guards from forging new access-keys, it falls short in providing complete privacy to users. This is because a malicious dealer or guard could collude and potentially relate user identities to the access-tokens issued, allowing them to observe the actions taken by users once they gain access to the resource. Therefore, additional measures are necessary to ensure full user privacy in such scenarios.

In order to protect user's privacy during the registration stage, it is required to break any traceability between users identities and users access-keys. To accomplish this, we use homomorphic cryptography to hide the relationship between identities and access-keys. In this section, we present a method that allows a user to securely hide the components of the access token as long as the discrete logarithm problem remains secure.

The resulting protocol for *anonymous registration and anonymous access* (ARA2) allows for the distribution of the registration among D dealers, while the access control will be carried out by a team of N guards. We note that in this protocol, security is based on the discrete logarithm problem. Thus, in order to prevent a coalition of users to use multiplicative properties to forge new access-keys, a redundancy function is included in the protocol. Algorithm 17 describes the whole protocol, but, for the sake of brevity, and without loss of generality, the use of the redundancy function has not been taken into account in the description of the protocol nor in Example 5.4.2. We will prove in Section 5.5.2 that, in this protocol, malicious dealers are not able to reveal the identity of the users nor forge new access tokens.

Algorithm 17 ARA2 Algorithm. Anonymous registration anonymous access.

```
1: System setup
2:     (a) Dealers agree on a prime $p$ and $m$.
3:     (b) Dealers agree a redundancy function $f$.     //e.g. a hash function
4:     (c) Each dealer computes its own degree $m_{d_i}$.
5:     (d) Each dealer decomposes its own degree $m_{d_i}$ and sends a component
       to each guard.
6:     (e) Every guard $g_j$ adds the received components to compute its own
       share $m^{g_j}$.
7: User identification
8:     (a) Each user generates two private values $s$ and $v$ such that $sv \equiv 1 \pmod{p-1}$.
9:     (b) Users generate a random value $r$ and set $x_u$ as the concatenation
       of $r$ and $f(r)$.
10:    (c) Users send his identification along with $pt_u = x_u^v$ (the masked $x_u$)
       to each dealer.
11:    (d) Dealers verify the identification credentials.
12:    (e) If the credentials are valid, each dealer $d_i$ replies to the user with:
           $pt_u^{m_{d_i}} \bmod p.$
13:    (e) Users can compute their credential (point) from the received points
       from the dealers as:
           $\langle x_u,\ y_u = \prod_{i=1}^{D} (pt_u^{m_{d_i}})^s \bmod p \rangle.$
14: Anonymous access
15:    (a) User sends his access-keys $\langle x_u, y_u \rangle$ to each one of the guards.
16:    (b) Guards check that $x_u$ capture the redundancy correctly.
17:    (b) Each guard computes $(x_u)^{m^{g_j}} \bmod p$.
18:    (c) Guards check if $y_u = \prod_{1 \le j \le N} (x_u)^{m^{g_j}} \bmod p$ .
19:    (d) Access is granted if the access-keys are valid.
```

The setup of the scheme implies the dealers to agree a prime p, as well as every dealer d_i to generate a random integer m_{d_i}. Then, each dealer partitions his integer m_{d_i} into N parts $m_{d_i}^{g^j}$ such that:

$$m_{d_i} = \sum_{j=1}^{N} m_{d_i}^{g^j},$$

and distribute each share to the guards through a secure channel. Thus, a guard g_j can compose an integer m^{g^j} as the sum of the received shares from the dealers:

$$m^{g^j} = \sum_{i=1}^{D} m_{d_i}^{g^j}$$

Note that, at the end of the distribution phase, both the set of dealers and the set of guards have different shares of the same secret integer m which is never stored anywhere and result from the sums:

$$m = \sum_{i=1}^{D} m_{d_i} = \sum_{j=1}^{N} m^{g^j}$$

Before the registration, each user generates a pair of private integers v and s such that $vs \equiv 1 \mod (p-1)$. Then, the user u selects an integer x_u and sends his identification together with a pretoken $pt_u = x_u^v \bmod p$ to each dealer.

If the identification is valid, then each dealer d_i can compute and reply to the user $pt_u^{m_{d_i}} \mod p$. Note that dealers do not have access to x_u, and therefore, no dealer can track the way the tokens are used by the users. We also note that, for big enough values of p, the probability that two users could generate the same x_u, which would lead two different users to have the same access token, is extremely low[3].

Once received a reply from all the dealers, the user can now compute his token $\langle x_u, y_u \rangle$ where:

$$y_u = \prod_{i=1}^{D} (pt_u^{m_{d_i}})^s \mod p.$$

[3]Negligible for p values of 1024 bits, which is nowadays a very conservative modular size.

Note that the process is such as it guarantees that:

$$y_u = \prod_{i=1}^{D} (pt_u^{m_{d_i}})^s \mod p = \prod_{i=1}^{D} ((x_u^v)^s)^{m_{d_i}} \mod p =$$
$$= x_u^{\sum_{i=1}^{D} m_{d_i}} \mod p \quad = x_u^m \mod p$$

where m is an agreed but unknown integer.

Note that, first, both v and s are private values, enrolled in masking/unmasking processes in an homomorphic cryptography framework; and, second, that the fact that the modulus p is a known prime is not a security issue because: both values v and s will remain secret for everyone but the user who generated them; and, $pt_u = x_u^v \bmod p$ is the only transmitted value which is not enough to reveal the hidden x_u value.

In the access phase the user provides his access token to each guard who can collectively check if it is valid. Example 5.4.2 illustrates the protocol taking into account artificially low setup values. We summarize the ARA2 protocol in Algorithm 17 and Example 5.4.2 illustrates the procedure.

Example 5.4.2. *Let us consider an scenario with two dealers ($D = 2$) and three guards ($N = 3$) to control the access of users to some resource. As in previous examples, let $p = 7919$.*

Let $m_{d_1} = 3401$ and $m_{d_2} = 1034$ be the (private) integers generated by the dealers. Each dealer partitions his integers into N shares and (securely) send them to the guards. For instance, let consider the partition of m_{d_1} as:

$$m_{d_1}^{g^1} = 1400 \quad m_{d_1}^{g^2} = 1001 \quad m_{d_1}^{g^3} = 1000$$

and the partition of m_{d_2} as:

$$m_{d_2}^{g^1} = 34 \quad m_{d_2}^{g^2} = 500 \quad m_{d_2}^{g^3} = 500$$

thuss, subsequently, the guards can obtain:

$$m^{g^1} = 1434; \quad m^{g^2} = 1501; \quad m^{g^3} = 1500$$

Consider a user willing to obtain an access token that randomly generates $x_u = 103$, $v = 7717$ and, $s = 1103$. Note that $sv \equiv 1 \pmod{p-1}$. All three *values are kept secret. The user then sends his identification together with the pretoken* $pt_u = x_u^v \bmod 7919 = 2976$ *to each dealer, who reply:*

$$pt_u^{m_{d_1}} \bmod p = 1646; \quad pt_u^{m_{d_2}} \bmod p = 5625$$

and the user now can compute his access token as:

$$\langle x_u = 103,\ y_u = 1646^{1103} \cdot 5625^{1103} \bmod 7919 = 4271 \rangle,$$

note that $m = 3401 + 1034 = 4435$*, and* $103^{4435} = 4271$*.*

Once the guards receive the access token, each one can work out the (partial) modular exponentiation from its integer share:

$$\begin{aligned} 103^{1434} \bmod 7919 &= 4898 \\ 103^{1501} \bmod 7919 &= 6001 \\ 103^{1500} \bmod 7919 &= 2211 \end{aligned}$$

and must collaborate in order to check whether the token is valid or not. Indeed, $4898 \cdot 6001 \cdot 2211 \bmod p = 4271 = y_u$*, and access should be granted.*

ARA2 protocol provides a fully decentralized and anonymous way to obtain access tokens. In Section 5.5.2 we prove that no coalition of guards and/or dealers and/or users, can reveal the identity of registered users; and, unless the discrete logarithm problem is solved, they are unable to forge new credentials.

5.5 Security Analysis

We devote this section to analyze the security properties of our schemes. In order to prove their unforgeability, against probabilistic polynomial time adversaries *Adv*, we propose a security game for each one of the protocols.

5.5.1 TRA2 Analysis

We devote this section to analyze the security properties of the TRA2 protocol. The security game is defined as:

- The *SysSetup* is run to get the polynomial $q(x)$, its partition into $\{q_i\}_{i \in N}$ polynomials, m, and p. The values m and p are given to *Adv*.

- Let $U \subsetneq G$ be a set of corrupt guards. Give $P = \{q_i\}_{i \in U}$ to Adv.
- Let $V \subseteq C$ be a set of corrupt users where $|V| < m$. Give the set of access-keys $K = \{Registration(id_i)\}_{i \in V}$ to Adv.
- Let $\mathcal{O}_{interp}(P, K)$ be the oracle that, by any method, considers the available information to interpolate the polynomial:

$$q(x) - \sum_{i \in U} q_i.$$

- Let $\mathcal{O}_{newk}(P, K)$ be the oracle that, by any method, considers the available information to obtains new pairs (a, b) such that $b = q(a)$.
- Let $\mathcal{O}_{succ}(p(x))$ be the oracle that returns 1 if $p(x) = q(x)$ and returns 0 otherwise.

Lemma 5.5.1 proves that, when conditions are met, our construction of the scheme is secure because adversaries cannot forge malicious credentials.

Lemma 5.5.1. *Provided that the number of users does not exceed m, and that not all the guards are malicious, TRA2 is unforgeable and no coalition of users and/or guards can forge valid access-keys.*

Proof. We note that there exist no method to interpolate the polynomial under the distributed control of the trusted guards (i.e., the result of $\sum_{i \notin U} q_i$). This is a fact regardless the computational power available to Adv. In the same way, the probability any call to $\mathcal{O}_{newk}$ to output a new valid access-key is also negligible, unless there were enough keys (points of the polynomial) that could eventually allow to interpolate $q(x)$.

We stress that the best the oracles can return, is a guess consistent with the available data. We note that there exists a combinatorial number of other many different guesses that are also consistent. Therefore, the probability of $\mathcal{O}_{succ}$ to return 1 is negligible. □

Despite the TRA2 protocol not employing encryption or signature methods, if the security conditions are met, there is no procedure to confirm the guesses made by an attacker, therefore TRA2 succeeds in providing security derived entirely from information theory, creating a system where partial information does not reveal anything about the scheme's secrets. Therefore, TRA2, provides Perfect Secrecy as defined in Sec. 2.5.3.

5.5.2 TDRA2 and ARA2 Analysis

We analyze in this section the security properties of TDRA2 and ARA2 protocols. We will name D the set of dealers, G the set of N guards that control the access to the resource, and C will denote the collective of users that apply for accessing to the resource. We consider that any combination of a subset of dealers, users and/or guards can collude to forge the protocol.

To analyze the unforgeability of TDRA2, for every probabilistic polynomial time adversary Adv, we propose a security game where:

- *SysSetup* is run to get m and p values which are given to Adv.
- *DealersSetup* is run by each dealer that, individually generate an m-degree polynomial mod p, and divide it into shares that safely send to each guard.
- Let $E \subsetneq G$ be a set of corrupt dealers. Give $R = \{q_i\}_{i \in E}$ to Adv.
- *GuardsSetup* is run by each guard that, individually combine all the polynomials received from the dealers to obtain its own m-degree polynomial mod p.
- Let $U \subsetneq G$ be a set of corrupt guards. Give $P = \{q_i\}_{i \in U}$ to Adv.
- Let $V \subseteq C$ be a set of corrupt users where $|V| < m$. Give the set of access-keys $K = \{Registration(id_i)\}_{i \in V}$ to Adv.
- Let $\mathcal{O}_{interp}(P, R, K)$ be the oracle that, by any method, consider the available information to interpolate either the polynomial:
$$q(x) - \sum_{i \in D} q_i,$$
or the polynomial:
$$q(x) - \sum_{i \in U} q_i.$$
- Let $\mathcal{O}_{newk}(P, R, K)$ be the oracle that, by any method, consider the available information to obtains new pairs (a, b) such that $b = q(a)$.
- Let $\mathcal{O}_{succ}(p(x))$ be the oracle that returns 1 if $p(x) = q(x)$ and returns 0 otherwise.

Lemma 5.5.1 proves that, when conditions are met, our TDRA2 protocol is unforgeable, because, no matter the computational power available to the adversaries, it is not possible to forge malicious credentials.

Lemma 5.5.2. *Provided that the number of users does not exceed m, and that neither all the guards, nor all the dealers are malicious, TDRA2 is unforgeable and no coalition of users, and/or dealers, and/or guards can forge valid access-keys.*

Proof. We note first that, after the dealers and guards setup, both collectives distributively guard the same secret polynomial $q(x)$. Second, that, individually, dealers and guards only have access to shares of $q(x)$. Third, that the shares from corrupt dealers/guards provided to *Adv* do not allow to interpolate $q(x)$, because, no matter the computational power, there exists no method to do it. Finally, that any number of access-keys provided to *Adv* do not change the scenario, unless the number of keys provided exceed the polynomial degree m. Thus, in any case, the best the oracles can return is a consistent guess with the available data of $q(x)$, but with no available method to check the plausibility of the guess against all the other many different guesses that, given any amount of information, are consistent. Therefore, the probability of $\mathcal{O}_{succ}$ to return 1 is negligible.

To sum up, regardless the computational power available to *Adv*, there exists no method to interpolate the polynomial under the control of the dealers and guards, and, therefore, it is not possible to forge new credentials unless the number of registered users reaches the grade of the polynomial, or all the dealers/guards collude to forge the system. □

We now analyze the security properties of ARA2 protocol. To do so, for every probabilistic polynomial time adversary *Adv*, we propose a security game where:

- *SysSetup* is run to get the m and p values that are given to *Adv*.
- *DealersSetup* is run by each dealer that, individually generate an exponent m_{d_i}, and divide it into shares that are safely sent to each guard.
- Let $E \subsetneq G$ be a set of corrupt dealers. Give $R = \{m_i\}_{i \in E}$ to *Adv*.
- *GuardsSetup* is run by each guard that, individually combine all the exponents received from the dealers to obtain its own exponent m^{g_j}.

- Let $U \subsetneq G$ be a set of corrupt guards. Give $P = \{m_i\}_{i \in U}$ to Adv.
- Let $V \subseteq C$ be a set of corrupt users. Give the set of access-keys $K = \{Registration(id_i)\}_{i \in V}$ to Adv.
- Let $\mathcal{O}_{disclog}(P, R, K)$ be the oracle that, by any method, consider the available information to solve the discrete logarithm problem.
- Let $\mathcal{O}_{newk}(P, R, K)$ be the oracle that, by any method, consider the available information to obtains a new pair (a, b) such that $b = a^m \bmod p$.

First, we prove in Lemma 5.5.3 that the protocol, as described, protects the users' identity.

Lemma 5.5.3. *ARA2 protocol ensures users' privacy.*

Proof. Let us stress that the access-key delivered to any user take into account a x_u value which, before sending them to the dealers, is masked using a (private exponent) modular exponentiation. Since the user is the only one who knows the operation to reverse this mask, all the possible values of x_u are equally probable for an Adv. Thus, it is not possible to relate a user identity to an access-key, even if an Adv intercepts all the dealers' responses to the user.

Guards receive no information about the identity of the users, and they grant access using exclusively the issued access-keys. Therefore, neither a subset of guards, nor a coalition of dealers and guards can track user identities. □

We now prove in Lemma 5.5.4 that no one can forge new access-keys.

Lemma 5.5.4. *According ARA2 protocol's description, the probability of forging new access-keys is negligible.*

Proof. The protocol states that a user access-keys is of the form:

$$\langle x_u,\ y_u = x_u^m \bmod p \rangle.$$

In order an adversary to forge a new access-key from an existing one, it would be necessary to partition x_u into two different components a_u and b_u, in order to, afterwards, derive the related a_u^m and b_b^m, that is:

$$\langle x_u = a_u b_u,\ y_u = (a_u b_u)^m \bmod p = a_u^m b_u^m \rangle.$$

We note that, since m is distributed and unknown, there is not enough information to factorize y_u to obtain a_u^m and b_u^m. Nevertheless, if eventually the adversary could carry out this process, it would be necessary, that at least two out of x_u, a_u and b_u, would consider the redundancy function f, which is highly unfeasible.

An adversary could try the following approach. Let him to call algorithm *Registration* for a chosen $x_u = ab$, and for $x_v = a^{-1}b$ as well. Given the respective values $y_u = (ab)^m \bmod p$ and $y_v = (a^{-1}b)^m \bmod p$, let the adversary to proceed as follows:

$$\begin{gathered}\langle x_u x_v \bmod p = b,\ (ab)^m (a^{-1}b)^m \bmod p\rangle \ \Rightarrow \\ \Rightarrow\ \langle b,\ (aba^{-1}b)^m \bmod p = b^{2m}\rangle\end{gathered}$$

from which it is possible to efficiently obtain $\langle b,\ b^m \bmod p\rangle$. Nevertheless, in order the access-keys to agree with the redundancy function, x_u, x_v and b should capture correctly the redundancy established by f, which again is highly unfeasible. □

Previous lemmas prove that, under ARA2 protocol, the identity of the users is preserved even in a post-quantum scenario, and that it is not possible for the partners to forge new access-keys as the discrete logarithm problem remains unsolved. Thus, we can conclude that ARA2 protocol is secure.

5.6 Time Complexity Analysis

We devote this section to provide a theoretic time complexity analysis of our identification protocols. We include in Appendix D a Proof of Concept implementation together with an empirical study of the time-complexity. As it is usual, we choose bit operation as the basic unit in our time-complexity analysis. Since all the operations are carried out modulus a prime p, the complexity of the operations will be expressed in terms of $\log p$. We recall Menezes et al. [1996] to the interested reader to inspect the complexity of modular operations.

It is important to note that the methods presented in the protocol require several steps and different sequential phases to operate effectively. However, we do not consider the complexity derived from auxiliary procedures, such as calls to the hash function, or the time devoted to communication between the parties. These functions typically have low complexity, and their use is not extensive. Comments on the influence of auxiliary and communication procedures are also included in Appendix D.

5.6.1 TRA2 and TDRA2 time complexity analysis

Due to the role of the polynomials in these protocols, we first stress that dealing with an m-degree polynomial does not imply to deal with $m+1$ terms. Security is not affected by considering disperse polynomials in which the number of terms is bounded by a constant t much lower than m (and, therefore, also much lower than p). We analyze the time complexity of the processes carried out by each partner once the parameters have been setup and the polynomials have been generated. Note that all these procedures can be carried out off-line and do not affect the complexity of the whole process.

First, regarding the figure of the dealers, for any given user registration request, each dealer d computes the result of his share of the polynomial, that can be computed with $\mathcal{O}(t \log^3 p) \approx \mathcal{O}(\log^3 p)$ bit operations. Second, regarding the users, once they have received all the messages from the dealers, they simply carry out an addition modulus p. A process with time complexity $\mathcal{O}(t \log p) \approx \mathcal{O}(\log p)$. Finally, regarding the guards, each one has to compute the result of his own share of the polynomial, with time complexity $\mathcal{O}(\log^3 p)$, and then collaboratively add the partial results of the rest of guards to grant or revoke the access, with time complexity $\mathcal{O}(n \log p) \approx \mathcal{O}(\log p)$. The whole identification process is a sequence of these procedures, therefore, for any single user requesting access, the complexity of the identification process is $\mathcal{O}(\log^3 p) + \mathcal{O}(d \log p) + \mathcal{O}(n \log^3 p) \approx \mathcal{O}(\log^3 p)$.

Let us note that, regarding the time complexity analysis, TRA2 can be considered as TDRA2 with $d = 1$. Note also that the number of bit operations scales linearly with the number of users in the scenario.

5.6.2 ARA2 time complexity analysis

Once again, we analyze the time complexity of the processes carried out by each partner once the parameters have been setup, and the polynomials have been generated.

Regarding the time complexity of the procedure carried out by ARA2 dealers, they only need to compute a single modular exponentiation, with complexity $\mathcal{O}(\log^3 p)$. Regarding the users, to mask the token implies a modular exponentiation, and, to reconstruct the access-key from the received shares implies the product of d values and a modular exponentiation, with time complexity $\mathcal{O}(d\log^2 p) + \mathcal{O}(\log^3 p) \approx \mathcal{O}(\log^3 p)$. Finally, each guard carry out a modular exponentiation to compute his partial result, with complexity $\mathcal{O}(\log^3 p)$, and then a set of n multiplications modulus p to combine all the partial results from all the guards in the collective, with complexity $\mathcal{O}(\log^3 p) + \mathcal{O}(n\log^2 p) \approx \mathcal{O}(\log^3 p)$.

Therefore, for any single user requesting access, the complexity of the whole identification process is $\mathcal{O}(\log^3 p) + \mathcal{O}(\log^3 p) + \mathcal{O}(\log^3 p) \approx \mathcal{O}(\log^3 p)$, and it is possible to see that the number of bit operations scale linearly with the number of users.

5.7 Applications

Our three protocols are application agnostic, meaning that they do not make any assumption or enforce any policy on how the final application will operate. Therefore, they are extremely portable, and together with their low time complexity, can be applied to a wide set of scenarios.

We here show how our anonymous access protocols can be used to create an untraceable blockchain airdrop, as well as an electronic voting scheme. We assume that we will be working with the ARA2 protocol for the anonymous registration property, but the presented procedures can be easily modified to consider the other schemes that we presented.

5.7.1 Blockchain Airdrop System

Airdrops have become a popular method in blockchain projects to increase engagement, popularity, and market capitalization by distributing free tokens to participants. However, ensuring that only legitimate and loyal users receive rewards is a challenge, as bots and fraudulent accounts can exploit airdrops. A common technique used to address this issue is the implementation of a whitelist, where interested participants provide their blockchain project experience and history for review by the airdrop organizers. However, this approach has two major limitations: it requires a manual review process, which can be time-consuming, and it compromises the anonymity of potential participants.

We propose a novel approach to address the challenges of airdrops using our ARA2 protocol and smart contracts as dealers. This approach preserves anonymity and involves participants proving their activity in the blockchain space by signing transactions that are old enough to qualify them as legitimate users.

1. Participant generates at random v and s such that $vs \equiv 1 \pmod{p-1}$. Where s acts as secret, and v as view factor of a masking scheme. Being p a large prime number.
2. Participant chooses an old and valid transaction t from its wallet history.
3. Participant signs a message containing t and the address of the airdrop. This will be used as identification $id = sign(t, address)$.
4. Participant chooses x_u at random and masks it as ${x_u}^v \bmod p$.
5. Participant sends the id and ${x_u}^v \bmod p$ to the airdrop address. The value is masked using v, preventing from other parties of the network to see the access-key returned later and front-running the user.
6. The smart contract verifies the identification's signature and the antiquity of t and stops if the signature is wrong or the transactions is not old enough.
7. The smart contract returns an access-key, computed using ${x_u}^v$ to the participant.
8. Participant recovers the access-key by applying the secret factor.
9. Participant sends the access-key to the airdrop organizers and a new receiving blockchain address t_n.
10. Organizers check the access-key and stop if false.
11. Organizers send tokens to t_n.

5.7.2 Electronic Voting Scheme

Electronic voting requires two properties that seem counterintuitive at first: to ensure that no double-voting occurs, while preserving elector's privacy. See Chapter 4 for more details.

We present a simple sketch of a possible voting system that achieves both privacy and democracy. We separate the registration and the voting processes, and it implies the existence of an identification authority (IA) that checks the membership of the electors to the census (and also provides the access-keys), and a remote polling station whose access is controlled by controllers that verify access-keys and count votes. The system can be modified to accommodate multiple identification authorities or use other anonymous access protocols that we have proposed in this Chapter.

1. Elector generates at random v, s such that $vs \equiv 1 \pmod{p-1}$.
2. Elector chooses x_u at random and masks it as ${x_u}^v \bmod p$.
3. Elector sends his personal identification and ${x_u}^v \bmod p$ to the IA.
4. The IA checks the identification belongs to a valid elector in the census. It stops if false.
5. The IA computes the access-key and returns it to the elector.
6. Elector sends his vote and access-key to the controllers that guard the remote polling station.
7. Remote polling station controllers check the validity of the access-key. They stop if false.
8. Controllers grant electors access to the ballot box, and, once the election is finished, count and publish the received votes.

5.8 Conclusions

We covered three anonymous access protocols. The schemes we propose decouple identification and the actions to carry once the access is granted. They are based on basic mathematical primitives and allow readily escalation. The protocols are lightweight and suitable to be implemented on any type of platform, and support a wide range of applications. We briefly showed how they could be adapted to support blockchain or e-voting applications.

The third protocol we propose, ARA2, removes the possibility of tracking the identity of a user of the service and the actions carried out once access to the service is granted. Under the other two protocols here proposed, the users identities are accessible only to a set of (assumed honest) authorities that are entitled to issue the access-keys that grant access to the resource. All the protocols we described allow immediate non-interactive registration, and, prevent users' multiple access. Two of them also provide post-quantum security. To the best of our knowledge, they are the first distributed registration schemes with these properties.

Compared to Agrawal et al. [2018], our proposals require a single round interaction for registration and another one for access, and the same mechanism enables the distributed registration and allows at the same time the distributed access. Thus allowing us to reduce the complexity of the protocol, making, at the same time, unnecessary the use of TOPRFs.

Regarding ACS, our protocols cannot be considered as such, since our protocols do not present a framework for connecting multiple organizations to users, nevertheless, our protocols share with ACS the goal of providing unforgeable, unlikable and anonymous access-keys. Besides, our protocols allow for one time access with minimal computational load, the whole process can be distributed, they do not require pseudonyms or multiple rounds of interaction, and, they provide post-quantum security.

Let us finally note, the three protocols offer different solutions adaptable to diverse scenarios. If post-quantum security is desired, then TRA2 or TDRA2 could be the option, while ARA2 is suitable when finding honest authorities is not feasible.

5.8.1 Future Work

The extension of these methods in order to allow each credential to be used more than one time is very interesting and we will study it in the future. In the same vein, allowing the revoking of credentials, or adding time limitations to them is also an interesting topic of research.

Another possible extension is to introduce some error-tolerance mechanism or failure support on the dealers/guards, because the protocols availability is affected when one of the dealers/guards is unreachable. An initial solution can be implemented considering separated sets of dealers/guards, that can be implemented to, either mirror every individual authority, or mirror the whole authority system. In both cases the time needed for the user should not be affected and the system should rapidly adapt to the inclusion or lost of guard entities.

Chapter 6

Conclusions

Hay libros cortos que, para entenderlos como se merecen, se necesita una vida muy larga.

Don Francisco de Quevedo

WEPZTBTCERFVGGADXIRI
VRXBLEEWGLWCKUSPSBEZ
GFQQIWWVPMEMGXTRURL
GMFTALAZNKON

One-time pad. k=Pero la comprensión de ciertos artículos se dejan como ejercicio para el lector, continuara

In this thesis we have explored some of the open problems of social collaboration as they were presented in Chapter 1. We covered along the chapters,4 and 5 how cryptography enables secure and trustless electronic voting and anonymous identification. These are two of the most relevant problems for distributed cooperation and many other problems can be reduced to instances of these dilemmas. Solutions based in well-established cryptographic protocols (see Chapter 2) and decentralized technologies (see Chapter 3) have been presented. Both schemes based in complexity and computability theories have been introduced under different assumptions to accommodate the maximum number of possible scenarios.

More specifically, we designed a voting scheme based on blind signatures in Section 4.2,that was later implemented as a Solidity smart contract in Appendix A. Leveraging blockchain technology and the privacy solutions of Monero, we designed a decentralized voting scheme in Section 4.3.1. This voting scheme was later expanded in Appendixes B and C with a theoretic description of the proposed blockchain and an implementation of the ring signature employed in the protocol. Finally, a voting scheme with perfect secrecy, and secure even in post-quantum scenarios was introduced in Section 4.4.

Regarding the identification problem, we described a problem where a set of guards safeguard a resource. The access to the resource is granted only if all the guards agree on the validity of the presented access-key. In this scenario, guards do not trust each other and only trust the verification process of the access-key. Three anonymous registration solutions for this problem were presented in Sections 5.3.1, 5.4 and 5.4.2. These protocols address the anonymous registration dilemma under different circumstances making use of polynomial interpolation, secret sharing schemes and partial homomorphic properties. No matter the scheme, the three protocols ensure that no access-keys can be forged. The protocols are later implemented to show their efficiency and feasibility in Appendix D.

We also stated possible lines of future work and extensions for each one of these topics. We also open-sourced all of our implementations, so other teams can benefit from them. Finally, let us note that we designed, implemented and analyzed protocols that exemplify how distributed cryptography can be used to build better, more reliable and robust solutions for our democratic societies. Ensuring the norm is respected, without degrading the privacy of the final users.

6.1 PhD Key Results

Here we schematically summarize the main academical results of our research work We differentiate between our two main lines of investigation: electronic voting and anonymous identification.

6.1.1 Electronic Voting

- A journal article titled "A two authorities electronic vote scheme", Larriba, Antonio M., José M. Sempere, and Damián López. Computers & Security 97 (2020): 101940.

- A journal article titled "Distributed Trust, a Blockchain Election Scheme", Larriba, Antonio M., et al. Informatica 32.2 (2021): 321-355.

- A journal article titled "SUVS: Secure Unencrypted Voting Scheme", Larriba, Antonio M., and Damián López. Informatica 33.4 (2022): 749-769.

- A journal article titled "A Solidity implementation of TAVS" Larriba, Antonio M., and Damián López. Frontiers in Blockchain 6: 10.

- An open source implementation of the ring signature scheme described in Section 3.3.2, and employed in the Distributed Trust scheme. The implementation, which shows the empiric time-complexity and scalability of these signatures, is described in more detail in Appendix B and can be freely consulted in https://github.com/Fantoni0/RingCTPerformance.

- An open source implementation of a Proof-Of-Concept that exemplifies TAVS can be implemented as a Solidity smart contract. The implementation is described in more detail in Appendix A and can be freely consulted in https://github.com/Fantoni0/svs.

- A Spanish patent application for SUVS with number P202131209.

6.1.2 Anonymous Identification

- A journal article titled "How to Grant Anonymous Access", Larriba, Antonio M., and Damián López. IEEE Transactions on Information Forensics and Security 18 (2022): 613-625.

- An open source implementation of a Proof-Of-Concept that exemplifies how these protocols can be implemented. The implementation is described in more detail in Appendix D and can be freely consulted in https://github.com/Fantoni0/ara2.

- A Spanish patent with substantive examination, and publication number ES2904423.

Bibliography

Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2005. doi: 10.1007/978-3-540-30580-4_6. URL `https://doi.org/10.1007/978-3-540-30580-4_6`.

Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4):79:1–79:35, 2018. doi: 10.1145/3214303. URL `https://doi.org/10.1145/3214303`.

Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In *Proceedings of the 2006 ACM Workshop on Privacy in the Electronic Society, WPES 2006, Alexandria, VA, USA, October 30, 2006*, pages 29–40, 2006. doi: 10.1145/1179601.1179607. URL `https://doi.org/10.1145/1179601.1179607`.

Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: password-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2042–2059. ACM, 2018. doi: 10.1145/3243734.3243839. URL `https://doi.org/10.1145/3243734.3243839`.

Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer, 2003. doi: 10.1007/978-3-540-40061-5_29. URL `https://doi.org/10.1007/978-3-540-40061-5_29`.

Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

David Arroyo, Jesus Diaz, and Francisco de Borja Rodríguez. Non-conventional digital signatures and their implementations - A review. In Álvaro Herrero, Bruno Baruque, Javier Sedano, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference - CISIS'15 and ICEUTE'15, 8th International Conference on Computational Intelligence in Security for Information Systems / 6th International Conference on EUropean Transnational Education, Burgos, Spain, 15-17 June, 2015*, volume 369 of *Advances in Intelligent Systems and Computing*, pages 425–435. Springer, 2015. doi: 10.1007/978-3-319-19713-5_36. URL `https://doi.org/10.1007/978-3-319-19713-5_36`.

Jbid Arsenyan, Gülçin Büyüközkan, and Orhan Feyzioglu. Modeling collaboration formation with a game theory approach. *Expert Syst. Appl.*, 42(4):2073–2085, 2015. doi: 10.1016/j.eswa.2014.10.010. URL `https://doi.org/10.1016/j.eswa.2014.10.010`.

Ahmed Ben Ayed. A conceptual secure blockchain-based electronic voting system. *International Journal of Network Security & Its Applications*, 9 (3):01–09, 2017.

Ahsan Aziz. Coercion-resistant e-voting scheme with blind signatures. In *Cybersecurity and Cyberforensics Conference, CCC 2019, Melbourne, Australia, May 8-9, 2019*, pages 143–151, 2019. doi: 10.1109/CCC.2019.00009. URL `https://doi.org/10.1109/CCC.2019.00009`.

László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of*

Computing, May 6-8, 1985, Providence, Rhode Island, USA, pages 421–429. ACM, 1985. doi: 10.1145/22145.22192. URL `https://doi.org/10.1145/22145.22192`.

Adam Back. Ring signature efficiency, 2015. Available at `https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684`.

Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 183–198. ACM, 2019. doi: 10.1145/3318041.3355458. URL `https://doi.org/10.1145/3318041.3355458`.

Igor Barinov, Viktor Baranov, and Pavel Khahulin. Poa network white paper. *URL: https://github. com/poanetwork/wiki/wiki/POA-Network-Whitepaper*, 2018.

Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, pages 274–283, 2001. doi: 10.1145/383962.384044. URL `https://doi.org/10.1145/383962.384044`.

Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2009. doi: 10.1007/978-3-642-03356-8_7. URL `https://doi.org/10.1007/978-3-642-03356-8_7`.

Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer*

Science, pages 139–155. Springer, 2000. doi: 10.1007/3-540-45539-6_11. URL `https://doi.org/10.1007/3-540-45539-6_11`.

Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582. Springer, 2001. doi: 10.1007/3-540-45682-1_33. URL `https://doi.org/10.1007/3-540-45682-1_33`.

Marianna Belotti, Nikola Bozic, Guy Pujolle, and Stefano Secci. A vademecum on blockchain technologies: When, which, and how. *IEEE Commun. Surv. Tutorials*, 21(4):3796–3838, 2019. doi: 10.1109/COMST.2019.2928178. URL `https://doi.org/10.1109/COMST.2019.2928178`.

Josh Benaloh, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, and Poorvi L. Vora. End-to-end verifiability. *CoRR*, abs/1504.03778, 2015. URL `http://arxiv.org/abs/1504.03778`.

Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 544–553, 1994. doi: 10.1145/195058.195407. URL `https://doi.org/10.1145/195058.195407`.

Jorge Bernal Bernabé, Martin David, Rafael Torres Moreno, Javier Presa Cordero, Sébastien Bahloul, and Antonio F. Skarmeta. ARIES: evaluation of a reliable and privacy-preserving european identity management framework. *Future Gener. Comput. Syst.*, 102:409–425, 2020. doi: 10.1016/j.future.2019.08.017. URL `https://doi.org/10.1016/j.future.2019.08.017`.

David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012. doi: 10.1007/978-3-642-34961-4_38. URL `https://doi.org/10.1007/978-3-642-34961-4_38`.

David Bismark, James Heather, Roger M. A. Peel, Steve Schneider, Zhe Xia, and Peter Y. A. Ryan. Experiences gained from the first prêt à voter implementation. In *First International Workshop on Requirements Engineering for e-Voting Systems, RE-VOTE 2009, Atlanta, Georgia, USA, August 31, 2009*, pages 19–28, 2009. doi: 10.1109/RE-VOTE.2009.5. URL `https://doi.org/10.1109/RE-VOTE.2009.5`.

Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001. doi: 10.1007/3-540-44647-8_13. URL `https://doi.org/10.1007/3-540-44647-8_13`.

Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013. doi: 10.1007/978-3-642-40041-4_23. URL `https://doi.org/10.1007/978-3-642-40041-4_23`.

Steven J. Brams and Peter C. Fishburn. *Approval voting, 2nd edition*. Springer, 2007. ISBN 978-0-387-49895-9.

Eric Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

John Burns and Chris J. Mitchell. Parameter selection for server-aided RSA computation schemes. *IEEE Trans. Computers*, 43(2):163–174, 1994. doi: 10.1109/12.262121. URL `https://doi.org/10.1109/12.262121`.

Gangshu Cai and Ned Kock. An evolutionary game theoretic perspective on e-collaboration: The collaboration effort and media relativeness. *Eur. J. Oper. Res.*, 194(3):821–833, 2009. doi: 10.1016/j.ejor.2008.01.021. URL `https://doi.org/10.1016/j.ejor.2008.01.021`.

Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 21–30. ACM, 2002. doi: 10.1145/586110.586114. URL `https://doi.org/10.1145/586110.586114`.

Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001. doi: 10.1007/3-540-44987-6_7. URL `https://doi.org/10.1007/3-540-44987-6_7`.

Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Blind signatures based on the discrete logarithm problem. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 428–432, 1994.

Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. *IACR Cryptol. ePrint Arch.*, 2014:429, 2014. URL `http://eprint.iacr.org/2014/429`.

Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996. doi: 10.1145/237814.238015. URL `https://doi.org/10.1145/237814.238015`.

Thomas E. Carroll and Daniel Grosu. A secure and anonymous voter-controlled election scheme. *J. Network and Computer Applications*, 32 (3):599–606, 2009. doi: 10.1016/j.jnca.2008.07.010. URL `https://doi.org/10.1016/j.jnca.2008.07.010`.

David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981. doi: 10.1145/358549.358563. URL `http://doi.acm.org/10.1145/358549.358563`.

David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203, 1982. doi: 10.1007/978-1-4757-0602-4_18. URL `https://doi.org/10.1007/978-1-4757-0602-4_18`.

David Chaum. Blind signature system. In David Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983*, page 153. Plenum Press, New York, 1983.

David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985. doi: 10.1145/4372.4373. URL `https://doi.org/10.1145/4372.4373`.

David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004. doi: 10.1109/MSECP.2004.1264852. URL `https://doi.org/10.1109/MSECP.2004.1264852`.

David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 118–167. Springer, 1986. doi: 10.1007/3-540-47721-7_10. URL `https://doi.org/10.1007/3-540-47721-7_10`.

David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 118–139, 2005. doi: 10.1007/11555827_8. URL `https://doi.org/10.1007/11555827_8`.

David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy*, 6(3): 40–46, 2008. doi: 10.1109/MSP.2008.70. URL `https://doi.org/10.1109/MSP.2008.70`.

Guomin Chen, Chunhui Wu, Wei Han, Xiaofeng Chen, Hyunrok Lee, and Kwangjo Kim. A new receipt-free voting scheme based on linkable ring signature for designated verifiers. In *2008 International Conference on Embedded Software and Systems Symposia*, pages 18–23. IEEE, 2008.

Lidong Chen. Access with pseudonyms. In Ed Dawson and Jovan Dj. Golic, editors, *Cryptography: Policy and Algorithms, International Conference, Brisbane, Queensland, Australia, July 3-5, 1995, Proceedings*, volume 1029 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995. doi: 10.1007/BFb0032362. URL `https://doi.org/10.1007/BFb0032362`.

Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020. doi: 10.1007/978-3-030-45721-1_26. URL `https://doi.org/10.1007/978-3-030-45721-1_26`.

Vikas Chouhan and Anshul Arora. Blockchain-based secure and transparent election and vote counting mechanism using secret sharing scheme. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19, 2022.

Sherman S. M. Chow, Siu-Ming Yiu, and Lucas Chi Kwong Hui. Efficient identity based ring signature. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 499–512, 2005. doi: 10.1007/11496137_34. URL `https://doi.org/10.1007/11496137_34`.

Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 372–382, 1985. doi: 10.1109/SFCS.1985.2. URL `https://doi.org/10.1109/SFCS.1985.2`.

S Barry Cooper. *Computability theory.* Chapman and Hall/CRC, 2017.

Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 72–83, 1996. doi: 10.1007/3-540-68339-9_7. URL `https://doi.org/10.1007/3-540-68339-9_7`.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5):481–490, 1997. doi: 10.1002/ett.4460080506. URL `https://doi.org/10.1002/ett.4460080506`.

Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation.* Cambridge University Press, 2015.

Jason Paul Cruz and Yuichi Kaji. E-voting system based on the bitcoin protocol and blind signatures. *IPSJ Transactions on Mathematical Modeling and Its Applications*, 10(1):14–22, 2017.

Kevin Curran. E-voting on the blockchain. *The Journal of the British Blockchain Association*, 1(2):4451.

Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer, 1988. doi: 10.1007/0-387-34799-2_26. URL `https://doi.org/10.1007/0-387-34799-2_26`.

Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology - EUROCRYPT 2001,* *International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 152–165, 2001. doi: 10.1007/3-540-44987-6_10. URL `https://doi.org/10.1007/3-540-44987-6_10`.

Ivan Damgård, Jakob Funder, Jesper Buus Nielsen, and Louis Salvail. Superposition attacks on cryptographic protocols. In Carles Padró, editor, *Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings*, volume 8317 of *Lecture Notes in Computer Science*, pages 142–161. Springer, 2013. doi: 10.1007/978-3-319-04268-8_9. URL `https://doi.org/10.1007/978-3-319-04268-8_9`.

John M. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8(3):253–259, 1984. doi: 10.1080/0161-118491859060. URL `https://doi.org/10.1080/0161-118491859060`.

Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994. doi: 10.1002/ett.4460050407. URL `https://doi.org/10.1002/ett.4460050407`.

Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989. doi: 10.1007/0-387-34805-0_28. URL `https://doi.org/10.1007/0-387-34805-0_28`.

Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures (extended abstract). In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469. Springer, 1991. doi: 10.1007/3-540-46766-1_37. URL `https://doi.org/10.1007/3-540-46766-1_37`.

Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering? In *2018 International Workshop on*

Blockchain Oriented Software Engineering (IWBOSE), pages 19–25. IEEE, 2018.

Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638. URL `https://doi.org/10.1109/TIT.1976.1055638`.

Yevgeniy Dodis. Shannon impossibility, revisited. In Adam D. Smith, editor, *Information Theoretic Security - 6th International Conference, ICITS 2012, Montreal, QC, Canada, August 15-17, 2012. Proceedings*, volume 7412 of *Lecture Notes in Computer Science*, pages 100–110. Springer, 2012. doi: 10.1007/978-3-642-32284-6_6. URL `https://doi.org/10.1007/978-3-642-32284-6_6`.

Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. Anonymous identification in ad hoc groups. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 609–626. Springer, 2004. doi: 10.1007/978-3-540-24676-3_36. URL `https://doi.org/10.1007/978-3-540-24676-3_36`.

Wolfgang Drechsler and Ülle Madise. Electronic voting in estonia. *Electronic voting and democracy: A comparative analysis*, pages 97–108, 2004.

Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985. doi: 10.1109/TIT.1985.1057074. URL `https://doi.org/10.1109/TIT.1985.1057074`.

Aleks Essex, Jeremy Clark, Richard Carback, and Stefan Popoveniuc. The punchscan voting system: Vocomp competition submission. *Proceedings of the First University Voting Systems Competition (VoComp)*, 2007.

Aleksander Essex, Jeremy Clark, and Urs Hengartner. Cobra: Toward concurrent ballot authorization for internet voting. In *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, Bellevue, WA, USA, August 6-7, 2012*, 2012. URL `https://www.us` enix.org/conference/evtwote12/workshop-program/presentation/essex.

W Ethereum. Ethereum yellowpaper. *Ethereum. URL: https://ethereum.github.io/yellowpaper/paper.pdf [accessed Sept, 2022]*, 2022.

Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987. doi: 10.1109/SFCS.1987.4. URL https://doi.org/10.1109/SFCS.1987.4.

David Ferraiolo, Janet Cugini, D Richard Kuhn, et al. Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. doi: 10.1007/3-540-47721-7_12. URL https://doi.org/10.1007/3-540-47721-7_12.

Hal Finney. Reusable proofs of work. *Web Archives Homepage*, 2004.

Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Optimal resilience proactive public-key cryptosystems. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 384–393, 1997. doi: 10.1109/SFCS.1997.646127. URL https://doi.org/10.1109/SFCS.1997.646127.

Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed rsa-key generation. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 663–672, 1998. doi: 10.1145/276698.276882. URL https://doi.org/10.1145/276698.276882.

Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005. doi: 10.1007/978-3-540-30576-7_17. URL `https://doi.org/10.1007/978-3-540-30576-7_17`.

Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019. URL `https://eprint.iacr.org/2019/953`.

Patrick Gallagher. Digital signature standard (dss). *Federal Information Processing Standards Publications, volume FIPS*, 186, 2013.

Évariste Galois and Peter M Neumann. *The mathematical writings of Évariste Galois*, volume 6. European mathematical society, 2011.

Shiyao Gao, Dong Zheng, Rui Guo, Chunming Jing, and Chencheng Hu. An anti-quantum e-voting protocol in blockchain with audit function. *IEEE Access*, 7:115304–115316, 2019. doi: 10.1109/ACCESS.2019.2935895. URL `https://doi.org/10.1109/ACCESS.2019.2935895`.

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007. doi: 10.1007/s00145-006-0347-3. URL `https://doi.org/10.1007/s00145-006-0347-3`.

Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold RSA for dynamic and ad-hoc groups. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2008. doi: 10.1007/978-3-540-78967-3_6. URL `https://doi.org/10.1007/978-3-540-78967-3_6`.

W. Morven Gentleman and G. Sande. Fast fourier transforms: for fun and profit. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '66 Fall Joint Computer Conference, November 7-10, 1966, San Francisco, California, USA*, volume 29 of *AFIPS Conference Proceedings*, pages 563–578. AFIPS / ACM / Spartan Books, Washington D.C., 1966. doi: 10.1145/1464291.1464352. URL `https://doi.org/10.1145/1464291.1464352`.

Jan Gerlach and Urs Gasser. Three case studies from switzerland: E-voting. *Berkman Center Research Publication No*, 3:2009, 2009.

Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33 (2):51–59, 2002.

Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 399–408. ACM, 1998. doi: 10.1145/276698.276852. URL `https://doi.org/10.1145/276698.276852`.

Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015. doi: 10.1145/2699436. URL `https://doi.org/10.1145/2699436`.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 203–225. ACM, 2019. doi: 10.1145/3335741.3335750. URL `https://doi.org/10.1145/3335741.3335750`.

Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 644–660. IEEE, 2020. doi: 10.1109/SP40000.2020.00048. URL `https://doi.org/10.1109/SP40000.2020.00048`.

Freya Sheer Hardwick, Apostolos Gioulis, Raja Naeem Akram, and Konstantinos Markantonakis. E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy. 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), pages 1561–1567. IEEE, 2018.

Juris Hartmanis and John E. Hopcroft. An overview of the theory of computational complexity. *J. ACM*, 18(2):444–475, 1971. doi: 10.1145/321650.321661. URL `https://doi.org/10.1145/321650.321661`.

Friorik P. Hjalmarsson, Gunnlaugur K. Hreioarsson, Mohammad Hamdaqa, and Gísli Hjálmtýsson. Blockchain-based e-voting system. In *11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018*, pages 983–986. IEEE Computer Society, 2018. doi: 10.1109/CLOUD.2018.00151. URL `https://doi.org/10.1109/CLOUD.2018.00151`.

Yang Hua-jie, Miao Xiang-hua, Zhu Hai-tao, and Li Yi-ran. Efficient certificateless ring signature scheme with identity tracing. *Information Security and Technology*, (7):9, 2014.

Kazuharu Itakura. A public-key cryptosystem suitable for digital multisignatures. *NEC J. Res. Dev.*, 71, 1983.

Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 162–177, 2000. doi: 10.1007/3-540-44448-3_13. URL `https://doi.org/10.1007/3-540-44448-3_13`.

Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 276–291. IEEE, 2016. doi: 10.1109/EuroSP.2016.30. URL `https://doi.org/10.1109/EuroSP.2016.30`.

Wen-Shenq Juang, Chin-Laung Lei, and Horng-Twu Liaw. A verifiable multi-authority secret election allowing abstention from voting. *Comput. J.*, 45(6):672–682, 2002. doi: 10.1093/comjnl/45.6.672. URL `https://doi.org/10.1093/comjnl/45.6.672`.

Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 37–63, 2010. doi: 10.1007/978-3-642-12980-3_2. URL `https://doi.org/10.1007/978-3-642-12980-3_2`.

David Kahn. *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet.* Simon and Schuster, 1996.

Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494. Springer, 2001. doi: 10.1007/3-540-44987-6_29. URL `https://doi.org/10.1007/3-540-44987-6_29`.

Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010. doi: 10.1007/978-3-642-14623-7_18. URL `https://doi.org/10.1007/978-3-642-14623-7_18`.

Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48 (177):203–209, 1987.

Neal Koblitz and Alfred J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptogr.*, 77(2-3):587–610, 2015. doi: 10.1007/s10623-015-0094-2. URL `https://doi.org/10.1007/s10623-015-0094-2`.

Koe, Kurt M. Alonso, and Sarang Noether. Zero to monero: Second edition. Available at https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf.

Nir Kshetri and Jeffrey M. Voas. Blockchain-enabled e-voting. *IEEE Softw.*, 35(4):95–99, 2018. doi: 10.1109/MS.2018.2801546. URL `https://doi.org/10.1109/MS.2018.2801546`.

Wei-Jr Lai and Ja-Ling Wu. An efficient and effective decentralized anonymous voting system. *CoRR*, abs/1804.06674, 2018. URL `http://arxiv.org/abs/1804.06674`.

Antonio M. Larriba, José M. Sempere, and Damián López. A two authorities electronic vote scheme. *Comput. Secur.*, 97:101940, 2020. doi: 10.1016/j.cose.2020.101940. URL `https://doi.org/10.1016/j.cose.2020.101940`.

Antonio M. Larriba, Aleix Cerdà-i-Cucó, José M. Sempere, and Damián López. Distributed trust, a blockchain election scheme. *Informatica*, 32 (2):321–355, 2021. doi: 10.15388/20-INFOR440. URL `https://doi.org/10.15388/20-INFOR440`.

Kibin Lee, Joshua I. James, Tekachew Gobena Ejeta, and Hyoung Joong Kim. Electronic voting service using block-chain. *J. Digit. Forensics Secur. Law*, 11(2):123–136, 2016. doi: 10.15394/jdfsl.2016.1383. URL `https://doi.org/10.15394/jdfsl.2016.1383`.

Jonathan Levin and Barry Nalebuff. An introduction to vote-counting schemes. *Journal of Economic Perspectives*, 9:3–26, 02 1995. doi: 10.1257/jep.9.1.3.

Chun-Ta Li, Min-Shiang Hwang, and Yan-Chi Lai. A verifiable electronic voting scheme over the internet. In *Sixth International Conference on Information Technology: New Generations, ITNG 2009, Las Vegas, Nevada, USA, 27-29 April 2009*, pages 449–454, 2009a. doi: 10.1109/ITNG.2009.93. URL `https://doi.org/10.1109/ITNG.2009.93`.

Chun-Ta Li, Min-Shiang Hwang, and Yan-Chi Lai. A verifiable electronic voting scheme over the internet. In Shahram Latifi, editor, *Sixth International Conference on Information Technology: New Generations, ITNG 2009, Las Vegas, Nevada, USA, 27-29 April 2009*, pages 449–454. IEEE Computer Society, 2009b. doi: 10.1109/ITNG.2009.93. URL `https://doi.org/10.1109/ITNG.2009.93`.

Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999. doi: 10.1007/3-540-46513-8_14. URL `https://doi.org/10.1007/3-540-46513-8_14`.

Epp Maaten. Towards remote e-voting: Estonian case. In Alexander Prosser and Robert Krimmer, editors, *Electronic Voting in Europe - Technology, Law, Politics and Society, Workshop of the ESF TED Programme together with GI and OCG, July, 7th-9th, 2004, in Schloß Hofen / Bregenz, Lake of Constance, Austria, Proceedings*, volume P-47 of *LNI*, pages 83–100. GI, 2004. URL `https://dl.gi.de/20.500.12116/29132`.

Philip D. MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2002. doi: 10.1007/3-540-45708-9_25. URL `https://doi.org/10.1007/3-540-45708-9_25`.

Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2111–2128. ACM, 2019. doi: 10.1145/3319535.3339817. URL `https://doi.org/10.1145/3319535.3339817`.

Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN 0-8493-8523-7. doi: 10.1201/9781439821916. URL `http://cacr.uwaterloo.ca/hac/`.

Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology -*

CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987. doi: 10.1007/3-540-48184-2_32. URL `https://doi.org/10.1007/3-540-48184-2_32`.

Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985. doi: 10.1007/3-540-39799-X_31. URL `https://doi.org/10.1007/3-540-39799-X_31`.

Peter L Montgomery. A survey of modern integer factorization algorithms. *CWI quarterly*, 7(4):337–366, 1994.

Ciara Moore, Máire O'Neill, Elizabeth O'Sullivan, Yarkin Doröz, and Berk Sunar. Practical homomorphic encryption: A survey. In *IEEE International Symposium on Circuits and Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014*, pages 2792–2795. IEEE, 2014. doi: 10.1109/ISCAS.2014.6865753. URL `https://doi.org/10.1109/ISCAS.2014.6865753`.

Eduardo Morais, Tommy Koens, Cees van Wijk, and Aleksei Koren. A survey on zero knowledge range proofs and applications. *CoRR*, abs/1907.06381, 2019. URL `http://arxiv.org/abs/1907.06381`.

Louis Joel Mordell. *Diophantine equations*. Academic press, 1969.

Rafael Torres Moreno, Jorge Bernal Bernabé, Antonio F. Skarmeta, Michael Stausholm, Tore Kasper Frederiksen, Noelia Martínez, Nuno Ponte, Evangelos Sakkopoulos, and Anja Lehmann. OLYMPUS: towards oblivious identity management for private and user-friendly services. In *2019 Global IoT Summit, GIoTS 2019, Aarhus, Denmark, June 17-21, 2019*, pages 1–6. IEEE, 2019. doi: 10.1109/GIOTS.2019.8766357. URL `https://doi.org/10.1109/GIOTS.2019.8766357`.

Teogenes Moura and Alexandre Gomes. Blockchain voting and its effects on election transparency and voter confidence. In *Proceedings of the 18th annual international conference on digital government research*, pages 574–575, 2017.

Francesc D. Muñoz-Escoí, Rubén de Juan-Marín, José-Ramón García-Escrivá, José Ramón González de Mendívil, and José M. Bernabéu-Aubán. CAP theorem: Revision of its related consistency models. *Comput. J.*, 62 (6):943–960, 2019.

Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

Moni Naor and Adi Shamir. Visual cryptography. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 1–12, 1994. doi: 10.1007/BFb0053419. URL `https://doi.org/10.1007/BFb0053419`.

Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, 1999. doi: 10.1007/3-540-48910-X_23. URL `https://doi.org/10.1007/3-540-48910-X_23`.

James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and Edward Roback. Report on the development of the advanced encryption standard (aes). *Journal of research of the National Institute of Standards and Technology*, 106(3):511, 2001.

Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019. doi: 10.1109/ACCESS.2019.2925010. URL `https://doi.org/10.1109/ACCESS.2019.2925010`.

Nicolas de Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendus à la pluralité des voix, 1785.

Harald Niederreiter. A public-key cryptosystem based on shift register sequences. In Franz Pichler, editor, *Advances in Cryptology - EUROCRYPT*

'85, Workshop on the Theory and Application of of Cryptographic Techniques, Linz, Austria, April 1985, Proceedings, volume 219 of *Lecture Notes in Computer Science*, pages 35–39. Springer, 1985. doi: 10.1007/3-540-39805-8_4. URL `https://doi.org/10.1007/3-540-39805-8_4`.

Shen Noether. Ring signature confidential transactions for Monero. *IACR Cryptol. ePrint Arch.*, 2015. Available at `https://eprint.iacr.org/2015/1098`.

Pierre Noizat. Blockchain electronic vote. In *Handbook of digital currency*, pages 453–461. Elsevier, 2015.

Hannu Nurmi. *Comparing voting systems*, volume 3. Springer Science & Business Media, 2012.

Ceyhun Onur and Arda Yurdakul. Electanon: A blockchain-based, anonymous, robust and scalable ranked-choice voting protocol. *CoRR*, abs/2204.00057, 2022. doi: 10.48550/arXiv.2204.00057. URL `https://doi.org/10.48550/arXiv.2204.00057`.

Pascal Paillier. Public-key cryptosystem based on discrete logarithm residues. *EUROCRYPT 1999*, 1999.

Sunoo Park, Michael A. Specter, Neha Narula, and Ronald L. Rivest. Going from bad to worse: from internet voting to blockchain voting. *J. Cybersecur.*, 7(1), 2021. doi: 10.1093/cybsec/tyaa025. URL `https://doi.org/10.1093/cybsec/tyaa025`.

Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59 (2):103–112, 2016.

Remigijus Paulavicius, Saulius Grigaitis, Aleksandr Igumenov, and Ernestas Filatovas. A decade of blockchain: Review of the current status, challenges, and future directions. *Informatica*, 30(4):729–748, 2019. URL `https://content.iospress.com/articles/informatica/inf1245`.

Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California,* *USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991. doi: 10.1007/3-540-46766-1_9. URL `https://doi.org/10.1007/3-540-46766-1_9`.

Adewole A Philip, Sodiya Adesina Simon, and Arowolo Oluremi. A receipt-free multi-authority e-voting system. *International Journal of Computer Applications*, 30(6):15–23, 2011.

Chinniah Porkodi, Ramalingam Arumuganathan, and Krishnasamy Vidya. Multi-authority electronic voting scheme based on elliptic curves. *I. J. Network Security*, 12(2):84–91, 2011. URL `http://ijns.femto.com.tw/contents/ijns-v12-n2/ijns-2011-v12-n2-p84-91.pdf`.

Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer, 1989. doi: 10.1007/0-387-34805-0_60. URL `https://doi.org/10.1007/0-387-34805-0_60`.

Tal Rabin. A simplified approach to threshold and proactive rsa. In *Annual International Cryptology Conference*, pages 89–104. Springer, 1998.

Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2003. doi: 10.1007/3-540-39200-9_32. URL `https://doi.org/10.1007/3-540-39200-9_32`.

Eric Rescorla. The transport layer security (TLS) protocol version 1.3. *RFC*, 8446:1–160, 2018. doi: 10.17487/RFC8446. URL `https://doi.org/10.17487/RFC8446`.

Ronald L Rivest. The threeballot voting system. 2006.

Ronald L Rivest and Warren D Smith. Three voting protocols: Threeballot, vav, and twin. *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)*, 2007.

Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. doi: 10.1145/359340.359342. URL `http://doi.acm.org/10.1145/359340.359342`.

Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman, editors, *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 164–186. Springer, 2006. doi: 10.1007/11685654_7. URL `https://doi.org/10.1007/11685654_7`.

Tim Roughgarden. Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559. *arXiv preprint arXiv:2012.00854*, 2020.

Tim Ruffing and Pedro Moreno-Sanchez. Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. *LNCS*, 10323:133–154, 2017. Proceedings of the International Conference on Financial Cryptography and Data Security.

Peter Y. A. Ryan. A variant of the chaum voter-verifiable scheme. In *Proceedings of the POPL 2005 Workshop on Issues in the Theory of Security, WITS 2005, Long Beach, California, USA, January 10-11, 2005*, pages 81–88, 2005. doi: 10.1145/1045405.1045414. URL `https://doi.org/10.1145/1045405.1045414`.

Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):1977–2008, 2020.

Donald G. Saari. *Geometry of voting*. Springer Science & Business Media, 2012.

Joel L Sachs, Ulrich G Mueller, Thomas P Wilcox, and James J Bull. The evolution of cooperation. *The Quarterly review of biology*, 79(2):135–160, 2004.

José Luis Salazar, Joan Josep Piles, José Ruíz-Mas, and José María Moreno-Jiménez. Security approaches in e-cognocracy. *Computer Standards & Interfaces*, 32(5-6):256–265, 2010. doi: 10.1016/j.csi.2010.01.004. URL `https://doi.org/10.1016/j.csi.2010.01.004`.

Roy Saltman. *The history and politics of voting technology: In quest of integrity and public confidence*. Springer, 2006.

Ravi S. Sandhu and Pierangela Samarati. Access control: principles and practice. *IEEE Commun. Mag.*, 32(9):40–48, 1994. doi: 10.1109/35.31284 2. URL `https://doi.org/10.1109/35.312842`.

Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 522–533. ACM, 1994. doi: 10.1145/195058.195405. URL `https://doi.org/10.1145/195058.195405`.

Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989. doi: 10.1007/0-387-348 05-0_22. URL `https://doi.org/10.1007/0-387-34805-0_22`.

José M. Sempere. Un sistema de identificación basado en un problema indecidible. In RS. González and C. Martínez, editors, *VII Reunión Española sobre Criptología y Seguridad de la Información (VII RECSI) (Oviedo, España)*, volume II, pages 793–803, 2002.

José M. Sempere. Un sistema de cifrado simétrico y algunas consideraciones sobre la seguridad computacional. In *VII Reunión Española sobre Criptología y Seguridad de la Información*, pages 131–137, 2004.

Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11): 612–613, 1979. doi: 10.1145/359168.359176. URL `http://doi.acm.org/10.1145/359168.359176`.

Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984. doi: 10.1007/3-540-39568-7_5. URL `https://doi.org/10.1007/3-540-39568-7_5`.

Michael I Shamos. Electronic voting-evaluating the threat. In *Third Conference on Computers, Freedom and Privacy, CPSR*, 1993.

Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949. doi: 10.1002/j.1538-7305.1949.tb00928.x. URL `https://doi.org/10.1002/j.1538-7305.1949.tb00928.x`.

Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2006. doi: 10.1007/11863908_2. URL `https://doi.org/10.1007/11863908_2`.

Joseph H Silverman and John Torrence Tate. *Rational points on elliptic curves*, volume 9. Springer, 1992.

Michael Sipser. Introduction to the theory of computation. *SIGACT News*, 27(1):27–29, 1996. doi: 10.1145/230514.571645. URL `https://doi.org/10.1145/230514.571645`.

Arkadii Slinko. *Algebra for Applications.* Springer, 2020.

Nick Szabo. Bit gold. *Recuperado de https://nakamotoinstitute. org/bit-gold/TVer página*, 2005.

Pavel Tarasov and Hitesh Tewari. Internet voting using zcash. *IACR Cryptology ePrint Archive*, 2017:585, 2017. URL `http://eprint.iacr.org/2017/585`.

Ruhi Taş and Ömer Özgür Tanrıöver. A systematic review of challenges and opportunities of blockchain for e-voting. *Symmetry*, 12(8):1328, 2020.

Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. doi: 10.1561/3300000030. URL `https://doi.org/10.1561/3300000030`.

Ai Thao Nguyen Thi and Tran Khanh Dang. Enhanced security in internet voting protocol using blind signature and dynamic ballots. *Electronic Commerce Research*, 13(3):257–272, 2013. doi: 10.1007/s10660-013-9120-5. URL `https://doi.org/10.1007/s10660-013-9120-5`.

José Luis Tornos, José Luis Salazar, Joan Josep Piles, Jose Saldana, Luis Casadesus, José Ruíz-Mas, and Julián Fernández-Navajas. An evoting system based on ring signatures. *Network Protocols & Algorithms*, 6(2): 38–54, 2014.

Patrick P. Tsang and Victor K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. *IACR Cryptology ePrint Archive*, 2004: 281, 2004. URL `http://eprint.iacr.org/2004/281`.

Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

Lingling Wang, Guoyin Zhang, and Chunguang Ma. A survey of ring signature. *Frontiers of Electrical and Electronic Engineering in China*, 3:10–19, 2008.

Wei Dai. B-money: anonymous, distributed electronic cash system. `http://www.weidai.com/bmoney.txt`, 1998. Online; accessed 22 February 2023.

Yifan Wu. An e-voting system based on blockchain and ring signature. *Master. University of Birmingham*, 2017.

Zhen Yu Wu, Ju-Chuan Wu, Sung-Chiang Lin, and Charlotte Wang. An electronic voting mechanism for fighting bribery and coercion. *J. Network and Computer Applications*, 40:139–150, 2014. doi: 10.1016/j.jnca.2013.09.011. URL `https://doi.org/10.1016/j.jnca.2013.09.011`.

Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Commun.*

Surv. Tutorials, 22(2):1432–1465, 2020. doi: 10.1109/COMST.2020.29697 06. URL `https://doi.org/10.1109/COMST.2020.2969706`.

Xuechao Yang, Xun Yi, Caspar Ryan, Ron G. van Schyndel, Fengling Han, Surya Nepal, and Andy Song. A verifiable ranked choice internet voting system. In *Web Information Systems Engineering - WISE 2017 - 18th International Conference, Puschino, Russia, October 7-11, 2017, Proceedings, Part II*, pages 490–501, 2017. doi: 10.1007/978-3-319-68786-5_39. URL `https://doi.org/10.1007/978-3-319-68786-5_39`.

Xuechao Yang, Xun Yi, Surya Nepal, Andrei Kelarev, and Fengling Han. A secure verifiable ranked choice online voting system based on homomorphic encryption. *IEEE Access*, 6:20506–20519, 2018. doi: 10.1109/ACCESS.2 018.2817518. URL `https://doi.org/10.1109/ACCESS.2018.2817518`.

Xuechao Yang, Xun Yi, Surya Nepal, Andrei Kelarev, and Fengling Han. Blockchain voting: Publicly verifiable online voting protocol without trusted tallying authorities. *Future Gener. Comput. Syst.*, 112:859–874, 2020. doi: 10.1016/j.future.2020.06.051. URL `https://doi.org/10.101 6/j.future.2020.06.051`.

Appendix A

A Solidity implementation of TAVS

TAVS, as defined in Section 4.2 assumes 2 non-colliding entities to ensure a secure electronic voting scheme. In some cases, it is possible to encounter scenarios where two entities involved in an election are unwilling or unable to share information. However, it is also true that such scenarios might be rare, and it may be impossible to find unrelated entities to participate in the election. The implementation detailed in this appendix reduces the assumption of honesty from two entities to one, without compromising the security properties of TAVS. This approach allows for greater flexibility in the selection of participants and can enable a wider range of entities to participate in the voting process. As a result,TAVS becomes more adaptable and better suited to meet the unique requirements of different elections.

One way to address the issue of finding two honest entities is to replace the tallying authority with an immutable smart contract. In doing so, the problem is resolved as smart contracts are self-governed entities that strictly adhere to the source code. Our solution satisfies these properties and has been made available for auditability, contributing to the open source community. While our implementation is fully equivalent to TAVS in all respects except for the tallying property, which is made public on the blockchain, allowing anyone to view the votes prior to the election's conclusion. To address this discrepancy between our implementation and the original proposal, we later present a solution that overcomes this issue.

A.1 From ECC to RSA

Because of its efficient computation, Ethereum is based, as well as Bitcoin, in the Secp256k1 [1] elliptic curve. It also supports some Bn254 [2] curve operations, as pre-compiles, because of its friendly pairing properties for zero-knowledge proofs. It is worth noting that no other cryptographic primitive currently has native or optimized support. As TAVS requires RSA for the blind signature scheme, it is imperative for us to address the implementation of the necessary support.

Solidity is a Turing complete language, so it is possible to implement any arbitrary system. However, Solidity is a high-level language and only supports a default word size of 32 bytes, which makes it difficult to implement direct support for big integers (integers that require more than 32 bytes to be represented) required in RSA and many other systems based on modular arithmetic. To handle big integers we employed arrays of bytes and assembly code in Yul [3]. Yul is an intermediate-level programming language that is designed to be compiled directly into low-level bytecode that can be used by the EVM. Its primary purpose is to facilitate the creation of assembly code for the EVM that can handle lower-level details. Yul serves a dual purpose: it allows for more detailed management of memory, and it can save gas by optimizing code at a lower level. By providing low-level control over memory management, Yul enables developers to create more efficient and optimized code that takes advantage of the unique characteristics of the EVM. Additionally, Yul's ability to optimize code at a lower level can lead to more cost-effective smart contract execution, ultimately benefitting the end-user.

To implement TAVS, we adapted the big number library developed by Firo [4] to be compatible with the latest Solidity versions. The developed library gives support for basic arithmetic operations using big integers as well as more complex operations that enable cryptographic primitives: modular exponentiation or computing inverses in a given modulo.

[1]Standard curve database|Secp256k1 https://neuromancer.sk/std/secg/secp256k1
[2]Standard curve database|Bn254 https://neuromancer.sk/std/bn/bn254
[3]Yul|Bytecode Language https://docs.soliditylang.org/en/latest/yul.html
[4]Github|Solidity Big Number https://github.com/firoorg/solidity-BigNumber

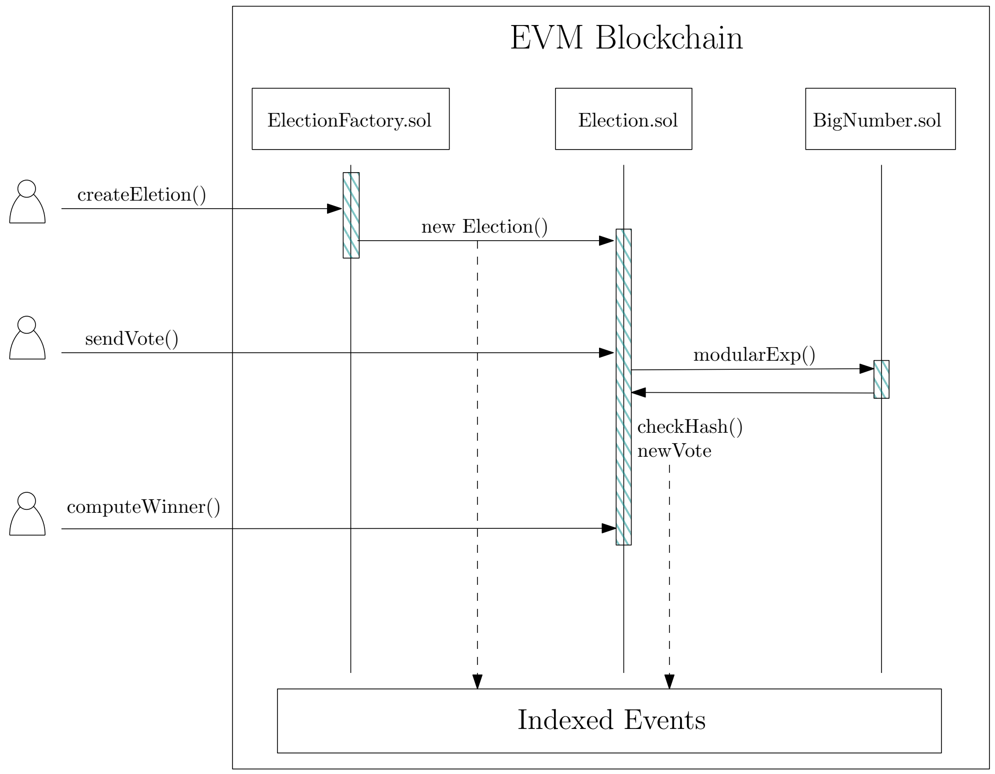


Figure A.1: Scheme representing the interaction between the user and the smart contracts.

A.1.1 Code Organization

The Solidity implementation of TAVS is publicly available in Github (https://github.com/Fantoni0/svs). A high level overview of the smart contract interaction can be found in Figure A.1. The smart contracts have been tested in a local EVM-compatible network and also have been deployed to a real testnet network. We used Mumbai testnet (a testnet to Polygon) because of the high cost of deploying directly to the main Ethereum network.

All the code has been developed using Solidity version 8.10[5] and the development library Hardhat[6]. Hardhat is one of the most complete Solidity libraries, it allows for compiling and running Solidity code locally, as well as it provides multiple helper functions for debugging smart contracts. The

[5] `https://docs.soliditylang.org/en/v0.8.10/`

[6] `https://hardhat.org/`

code is structured as follows:

- `contracts`: Contains all the smart contracts that constitute the implementation of TAVS. They are smart contracts written in Solidity.
  - `BigNumber.sol`: A Solidity library adapted from the implementation by Fire [7]. Contains all the code needed to deal with big integers and implement the blind signature scheme based on RSA.
  - `ElectionFactory.sol`: A smart contract that implements the factory design pattern. It creates and deploys instances of Election contracts. Handles and archives the created elections. It emits a event `NewElection` when a new Election is created. The Mumbai network deployment of this contract can be found in address 0xEBa9F87654171f88004f519CC18EfBD8A02e9421.
  - `Election.sol`: It contains all the logic to implement a TAVS election. Handles candidates, verifies votes and stores the current state of the election. It emits a `NewVote` event when a new Vote is received. The Mumbai network deployment of this contract can be found in address 0x9E459651D2A14B100a310FDd542954bd9565dFC0.
- `deploy`: Contains files to deploy the contracts in networks outside local deployment.
  - `Deploy.ts`: Deploys the ElectionFactory smart contract to a specified network. It creates an Election and sends 10 random votes.
  - `*Template.ts`: A set of template files the interested reader can use to carry out his own election process. See Sec. A.4 for a detailed guide.
- `verify`: Contains the file `arguments.js`, which contains a description of the parameters of the Election smart contract constructor needed to verify the contract. Verifying contracts allows to upload the source code to blockchain explorers such as Etherscan.
- `scripts`: Auxiliary files that implement different functionalities.

[7]Solidity Big Number - https://github.com/firoorg/solidity-BigNumber

  - `Tavs.ts`: Utility functions to simulate the IA in TAVS. it also generates random and valid votes for the scheme.
  - `Utils.js`: Different functions needed for testing: packing parameters, list functions etc.

- `test`: Contains typescript tests to verify the integrity of the smart contracts. See Sec. A.2 for more details.

A.2 Tests

Ethereum is a global adversarial network that operates on economic incentives. This means that any bug or vulnerability in the code can be exploited by attackers to obtain an economic reward. Moreover, these types of attacks do not require physical access and allow the attacker to remain anonymous. Therefore, testing the code is of utmost importance when developing smart contracts. The folder `test` contains various tests that analyze the behavior of the code. To run them and verify the validity of the code, the user can simply run `npx hardhat test`. We present here a short list of these assessments and its expected result.

- `ElectionFactory.ts`.

  1. Creates a valid election. It should create a new election and trigger the `NewElection` event.
  2. Creates an invalid election. The test should fail and the transaction reverts with error message "No elections shorter than 1 hour allowed".
  3. Creates an invalid election. The test should fail and the transaction reverts with error message "No elections longer than 4 days allowed".

- `Election.ts`.

  1. Creates a valid vote. It should create and send a new vote and trigger the `NewVote` event.
  2. Creates an invalid vote with the wrong hash. The test should fail and the transaction reverts with error message "Invalid hash".

3. Cends a vote after the election is finished. The test should fail and the transaction reverts with error message "Election has already finished. No more votes accepted".
4. Chould compute the winner of an election. The test sends a unique vote, forces the election to end and checks the winner is the voted candidate.
5. Tries to compute the winner of an election before the election is finished. The test should fail and the transaction reverts with error message "Election must be finished to compute tally."
6. Simulates and election with multiple votes and proceeds to compute the winner of the election.

A.3 Properties

Our Solidity implementation satisfies all the properties presented by TAVS. Moreover, our implementation does not compromise the quality of the voting system. In fact, some properties of our approach benefit from its immutability and decentralization. In the following section, we provide a brief overview of the properties of the TAVS election scheme, and explain how our implementation satisfies them. For a detailed demonstration of these properties, please refer to Section 4.2, where they are explicitly stated. Here, we formalize the properties of our implementation in the form of lemmas. Note that we will not provide a proof for each lemma whenever the proof is a direct consequence of the arguments presented.

Lemma A.3.1. *TAVS is private, and therefore, it is not possible to relate a vote with the elector who casted it.*

Our implementation not only guarantees voters' privacy, it also improves the degree of privacy provided in TAVS. TAVS' privacy is derived from the assumption of two honest non-colliding entities. Since we substitute the RPS with a smart contract, this assumption is no longer needed. At first, the idea of privacy and a public blockchain may seem conflicting. Nonetheless, please note that ballots sent to the RPS (the smart contract in our implementation), are not linked to the elector's identity in any form. Once the ballot has been signed by the IA, the elector can generate a burner address, not linked to her in any way, and send the ballot. The validity of the ballot depends on the digital signature by the IA. Hence, we can benefit from the public verifiability of the blockchain without degrading privacy.

Lemma A.3.2. *TAVS guarantees the Integrity of the vote, thus it is unfeasible for any partner in the system to modify a ballot without detecting the forgery.*

Lemma A.3.3. *TAVS ensures the Correctness of the final tally since it only considers verified and correct ballots.*

Lemma A.3.4. *TAVS provides Verifiability since any elector in the census can verify that her vote has been taken into account in the way it was casted.*

Verifiability is hold when all the processing on ballot in order to go through the process do not affect to the ballot itself. In the implementation we propose, the logic of the processing is described in a smart contract, and the results are published in a globally distributed network with thousands of participants. Thus, the implementation of the protocol is more resilient to attacks (as, for instance, DDoS attacks) than a clasical implementation that considers a private-access public bulletin board. Therefore, the verifiability process of the ballots and their integrity can be publicly audited, guaranteeing that only correct votes are accepted.

Lemma A.3.5. *TAVS is a Democratic scheme since only electors in the census can vote.*

Lemma A.3.6. *TAVS guarantees the Uniqueness of the votes by ensuring that electors can only vote once.*

Because our implementation maintains the IA and the blind signature scheme, we provide the same democracy and uniqueness as TAVS. The registration procedure does not differ from TAVS. Hence, registered electors can only vote once, since they only have one signed ballot, and, only electors in the census are able to get the signed ballot.

The only difference with respect to TAVS is the immediateness of the tally. In TAVS the final tally is computed and only revealed at the end of the election, since it depends on the authority that plays the role of Remote Polling Station. In our implementation, everything is in the blockchain, hence it is public by default. This means that everyone can see partial votes and compute a partial tally even before the election is finished.

Despite this can be considered not as an issue, it is usually considered as one. To address this (potential) drawback, we note that votes can be encrypted within a public key cryptosystem, whose key is set before hand by the IA (or alternatively by a set of parties), broadcasting the public key together with the election parameters, and revealing the decrypting key only after the end of the election. If in some scenarios the IA cannot be trusted with guarding this key. In those cases, a threshold system could be employed, in such a way that the responsibility of aggregating the key, once the election is finished, rests on a set of reputed parties. For that end, a threshold RSA system Rabin [1998], Damgård and Koprowski [2001] can be used. The parties guarding the keys must hold two requirements: they have to be interested in the correct development of the election; and, they must have antagonistic interests. The first ensures the honest participation of the parties, and the second one prevents malicious collaborations between them. Hence, political parties and/or a subset of electors could conform a suitable the set of guarding parties. This threshold scheme can be configured on demand to tolerate potential errors in the key recovering phase. So that if some parties are unable or unwilling to participate, a subset of honest parties can still decrypt the votes and carry on with the election. Please note, that this would only ensure the anonymity of the tally until the election ends. Furthermore, the privacy of users would not be affected in any form.

A.4 How to create your own election

In this section we show how to leverage our implementation and the open sourced code to create a particular election. The deployment and interaction will be created in the Mumbai network to reduce gas fees. This tutorial assumes the reader to have an up to date `Node.js` version installed.

1. Clone and install the code.

   ```
   git clone https://github.com/Fantoni0/svs && cd svs/
   npm install
   ```

   To compile and verify the smart contracts simply run:

   ```
   npx hardhat test
   ```

2. Get an address and funds.

   To operate in the blockchain environment it is necessary to have an EOA address. Ownership of addresses is determined by the associated secret key. It is possible to use services such as Vanity-Eth[8] or Crypterium[9] to generate your own key. Once the key is available, it is necessary to add it in the `env/.env` file as `MUMBAI_PRIVATE_KEY`. The next step is to uncomment the lines adding the Mumbai network in `hardhat.config.ts` file. This will let hardhat use the key to send the transactions.

   Once the key is ready, it is mandatory to get some funds available to pay for the transactions gas. Polygon Faucet[10] is a service that gives away small amounts to allow developers to pay for some transactions. To do so it is necessary to paste our address and then claim the funds.

3. Generate the keys to simulate the IA.

   The open-source implementation already includes a pair of public and private RSA keys that simulate the IA. This is sufficient for test purposes but a new pair will be needed to generate secure elections. It is possible to do it using OpenSSL:

   ```
   openssl genrsa 2048 -out private-key.pem
   openssl rsa -pubout -out public-key.pem -in private-key.pem
   ```

   From the output it is possible to extract the hexadecimal representation from the generated keys and substitute the modulo, public and private key instances in the code.

4. Create a particular Election Factory. (Optional)

   There is already an instance of the Election Factory, but a new one can be deployed by running:

   ```
   npx hardhat deploy --network mumbai --tags ElectionFactory
   ```

[8] `https://vanity-eth.tk/`
[9] `https://mycrypto.tools/ethaddress.html`
[10] `https://faucet.polygon.technology/`

If generated, the new address must be copied in the smart contract to be deployed. We need the Election Factory address to be able to interact with the contract and create new elections in the future.

5. Create an instance of Election.

   It is possible to make use of the file: `deploy/electionTemplate.ts` and make the necessary changes to adjust the election to our needs and run:

   ```
   npx hardhat deploy --network mumbai --tags Election
   ```

   As done previously, we will need to copy the address in which the Election smart contract was deployed. We'll need the address to send the future votes.

6. Vote.

   To finally send a vote, simply set the vote in `deploy/electionTemplate.ts` and execute:

   ```
   npx hardhat deploy --network mumbai --tags Vote
   ```

7. Verify your contract. (Optional)

   If it is desired to verify the deployed contracts, we need to obtain some API keys. First the `etherscan_api_key` in `hardhat.config.ts` must be specified. Second, the ElectionFactory program can be simply verified by providing the address in which it was deployed.

   ```
   npx hardhat --network mumbai verify ADDRESS_TO_VERIFY
   ```

   To verify the Election contract, since it was called with arguments, it is mandatory to provide the exact same arguments in the deploy/arguments.js, otherwise the verification will fail. To check for the parameters is is possible to use the Mumbai Block Explorer.

   ```
   npx hardhat verify --network mumbai --constructor-args
   verify/arguments.js ADDRESS_TO_VERIFY
   ```

A.5 Gas analysis: Costs of having an election

For the rest of this section, we assume the current prices of ETH (1410$) and MATIC (0.90$) at the moment of writing. Fees are paid in the currency of the network. Hence, if we operate in the Mumbai testnet, fees will be paid in MATIC, and in ETH if the deployment is made in the Ethereum mainnet. In Table A.1 we depict the gas units and USD costs of running the contracts and interacting with them in the Mumbai and Ethereum network respectively. The experiments have been carried out 200 times and the results have been averaged. Table represents the single execution cost of a given method.

All the tests have been run with the same `Hardhat` framework used for development. The plugin
`hardhat-gas-reporter` provides a detailed report on the gas metrics used in the tests. As covered in Section A.2, by executing `npx hardhat test`, the user can test the implementation itself, and replicate the presented gas cost analysis. Small variances depending on the average price per gas unit may occur.

Contract	Method	Gas Units	USD Cost (Mumbai)	USD Cost (Ethereum)
Election	computeWinner	$68,000$	0.0024$	4.22$
Election	sendVote	$39,717$	0.015$	2.47$
ElectionFactory	createElection	$235,067$	0.01$	14.56$
ElectionFactory	Deployment	$4,209,467$	0.16$	260.81$

Table A.1: Average costs of execution in the Mumbai and Ethereum network. Average price per gas unit of 43 gwei.

As shown in Table A.1, the cost of running the election process in Mumbai is significantly cheaper than carrying out the same process in Ethereum. The costs for all methods, regardless of the network, show a linear dependency with their complexity. The more computation and storage required, the higher the costs. Deploying the ElectionFactory contract is especially costly since it needs to upload the entire contract into the blockchain.

All methods listed in Table A.1 have constant costs except for the `computeWinner` method. This method depends on the internal state of the contract and needs to iterate over the list of all candidates to compute the most voted one. The length of the candidate list affects the computation and, consequently, the gas units and associated costs. Since the previous table only reports average units over the run of multiple tests, the effect of the candidate list on the `computeWinner` method is not captured. For this reason, we provide a specific description of the cost depending on the cardinality of the candidate's list in Table A.2. We isolated the cost of calling the method from other tests to avoid any possible cross-contamination of results. This table shows that the gas units and associated costs increase linearly with the length of the candidate list. Therefore, it is important to consider the size of the candidate list when estimating the cost of running the `computeWinner` method.

Number of candidates	Gas Units	USD Cost (Mumbai)	USD Cost (Ethereum)
2	$117,179$	0.004$	7.10$
4	$129,393$	0.005$	7.84$
8	$167,397$	0.006$	10.14$
16	$195,889$	0.007$	11.87$
32	$307,184$	0.011$	18.62$

Table A.2: Costs for `computeWinner` method. Average price per gas unit of 43 gwei.

As we can see in Table A.2, while the costs of computing the winner of the election increase with the number of candidates, it is not a severe change. Specially if we consider that elections with 32 or more candidates are unlikely, or compare the number of gas units with the deployment of the smart contract reflected in Table A.1.

Results show that, using our implementation of TAVS is perfectly feasible and affordable even for large elections. The deployment of TAVS in networks such as Mumbai is affordable and within reach for everyone, and the use of Ethereum, although more expensive, is still an effective solution. Moreover, the costs associated with TAVS are much lower than those related to running a traditional election. Overall, these results demonstrate the practicality and cost-effectiveness of TAVS as an e-voting system.

Appendix **B**

A Benchmark for Ring Signatures

In this Appendix, we present our implementation to craft and verify RingCT (see Section 3.3.2) signatures. The purpose for this is twofold: show the feasibility of this approach and obtain real empiric values to asses the complexity of Distributed Trust.

In contrast to modular exponentiation, ring signatures cannot be considered as a single operator. Ring signatures comprise multiple basic operators, making the comparison with modular exponentiation non-trivial. The computational time complexity of ring signatures is predominantly dominant due to their complexity. Furthermore, the time complexity of ring signatures depends on several parameters besides the input size, including the size of the ring and the desired level of security. For this reason, in Section 4.3.3, we kept the computational complexity associated with crafting/verifying a signature as a constant variable to offer a clear perspective of the time complexity. Nonetheless, we have now provided an implementation of these signatures to present an empirical result of the actual transactions per second (TPS) of the system.

To analyze the performance of ring signatures, we developed a Python 3 based code, which is publicly accessible on GitHub[1]. We have built a framework to test the impact of different parameters and elliptic curves on the performance of ring signatures. Our aim with this framework is to offer a practical implementation and obtain actual performance time measurements.

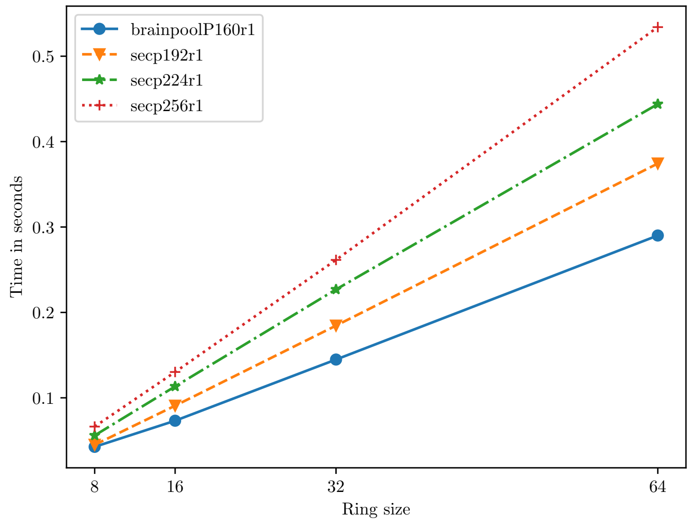
Figures B.1a and B.1b demonstrate the elapsed time required to craft or verify a signature respectively, for different parameters. We have compared

[1]`https://github.com/Fantoni0/RingCTPerformance`

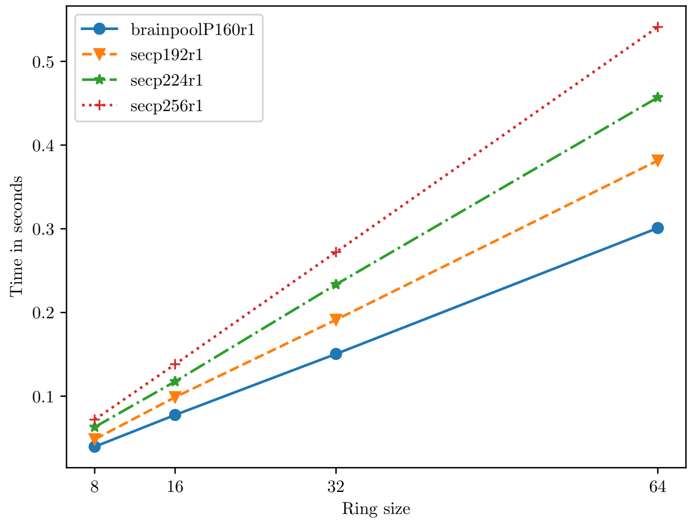
four different elliptic curves in this analysis, each providing a distinct level of security. BrainpoolP160r1 provides 80 bits of security; Curve-192 provides 91 bits; Curve-224 provides 112 bits; and, Curve-256 provides 128 bits. The Brainpool curve offers a security level equivalent to using RSA with a 1024-bit modulo Gallagher [2013], which is more than sufficient to safeguard the voter's privacy[2]. As expected, the time required for signature crafting or verification increases linearly with the size of the ring, or the complexity of the curve. We believe that utilizing brainpool and a ring size of 64 public keys is more than adequate to achieve the required security for the voting scheme of Distributed Trust.

The figures presented here were obtained using a personal computer: AMD Ryzen 7 3700X with 16 threads. As there are no dependencies between transactions, verification is a parallelizable task. We take advantage of this by using multiple cores. Using a professional server's processor and decentralizing the task among multiple servers would yield a great performance improvement. Nonetheless, a single personal computer is capable of verifying 3-4 TPS and 200 in a minute, even when using excessively high standard security parameters.

[2]A 1024-bit RSA key is deemed safe for the next few decades. No key larger than 829 bits has ever been factored.



(a) Time required to craft a ring signature under different ring sizes and elliptic curves.



(b) Time required to verify a ring signature under different ring sizes and elliptic curves.

Figure B.1: Ring signature performance times for crafting and verifying the same message under different parameters.

Finally, it is worth noting that the times for crafting and verification are very similar because the verification algorithm, described in Section 3.3.2, requires recreating the signature to check its validity.

Appendix C

Distributed Trust Technical Specification

This appendix aims to provide the technical specifications of the blockchain architecture outlined in the Distributed Trust voting scheme described in Section 4.3.1. The objective is to provide a comprehensive overview of the required structure and specifics necessary for the implementation of the blockchain required to execute the voting protocol.

The first section details the essential data structures that will define our blockchain, including blocks and transactions.

The second section outlines the necessary methods that the blockchain nodes will need to run to establish a functional blockchain. It is important to note that certain implementation details, such as node discovery, size in bytes, and network particulars, are outside the scope of this appendix. Nonetheless, we will provide enough information for interested readers to construct a functional blockchain that supports the Distributed Trust voting scheme.

C.0.1 Blockchain data structures

In our blockchain architecture, we define two fundamental data structures: transactions and blocks. Our blockchain protocol is designed for electronic voting, and as such, there is no need for monetary tokens. However, we have decided to provide support for a token to enhance the flexibility of the blockchain.

During the registration phase, parties must provide the $OTPK$s (see Section 3.3.1) with enough tokens to create transactions. Sufficient tokens must be assigned beforehand to allow the elector to vote multiple times, thereby preventing coercion.

Transactions

Transactions are the basic unit of information on the blockchain. They are broadcasted into the public network and added to the pool of pending transactions until they are added into a block. Transactions, in the UTXO model, are defined by their set of inputs and outputs. In Distributed Trust, transactions are as a communication mean to register votes. Inputs represent the elector's $OTPK$, and outputs represent the parties' (candidates) addresses. The elector decides the direction of her vote by configuring the outputs of the transaction. Table C.1 discloses the structure of transactions, and Table C.2 shows the structure of inputs and outputs.

Field	Definition
Version	Number version of the protocol
Inputs	List of referenced inputs
Outputs	List of outputs
Vote	Encrypted elector's vote
Signature	Ring signature of the transaction
Transaction ID	Hash identifying the transaction

Table C.1: Transaction structure.

Field	Definition
Amount	Amount of tokens in the key
Address	Public key identifying the address

Table C.2: Input and output structure.

Blocks

A block is a set of ordered and validated transactions. Blocks constitute the basic building component of a blockchain. Table C.3 shows the structure of an ordinary block in Distributed Trust.

The block header is used to verify the integrity of the block, while the transactions are the actual data stored in the blockchain. The previous block hash is used to link the current block to the previous block, creating the chain of blocks. The Merkle root is a hash of all the transactions in the block, which allows for efficient verification of transaction inclusion in the block. The nonce is a value that is adjusted by miners to satisfy the difficulty target, which is a measure of how difficult it is to find a block hash that satisfies the current network's consensus rules.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Signature	Digital signature of the block creator
Transactions	List of the transactions contained
Merkle Root	Merkle's root hash of the transactions
Block ID	Hash identifying the block

Table C.3: Block structure.

The configuration blocks are used to store the public parameters and cryptographic keys that are needed to run the election, and they are immutable once created. The first configuration block (Table C.4) contains the basic parameters, such as the number of parties, their public keys, their private key commitments, and the RSA parameters used in the election. The second configuration block (Table C.5) contains the starting and ending times of the election and the valid RingCT parameters that can be employed.

The last tallying block (Table C.6) is created after the voting phase is over, and it contains the encrypted votes and the secret key needed to decrypt them. It also includes a summary of the election results, such as the total number of votes cast and the number of votes received by each party.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Height	Distance to the first block
RSA parameters	Public key v and modulus n
Private shares	Commitments g^{s_i} of the private keys for each party
Public keys	Parties' public keys
Block ID	Hash identifying the block

Table C.4: First block in the chain describing configuration parameters.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Start Time	Start time of the election
End Time	End time of the election
Ring Size	Minimun accepted ring size for ring signatures
Options	List of options to vote for in the election
$OTPK$s	List of $OTPK$s and their corresponding random number R
Block ID	Hash identifying the block

Table C.5: Second block in the chain describing configuration parameters.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Secret key	Recovered secret key to decrypt votes
Results	Final election tally
List of results	Referenced transactions for each result
Block ID	Hash identifying the block

Table C.6: Last block in the chain describing the tally results.

In Figure C.1, we can see the structure of a complete blockchain. We can appreciate how the different blocks work, and how the different data structures are related to build the blockchain.

C.0.2 Methods

This section presents a set of algorithms that define the methods that blockchain nodes must follow to operate effectively. These methods provide a comprehensive overview of the various scenarios that nodes may encounter, including processing new blocks, handling transactions, and following the longest chain. By following these algorithms, nodes can ensure that they are operating efficiently and effectively in the blockchain network.

In the following algorithms, we make the assumption that certain functions are already defined. For instance, the hash() function generates a hash for a given input, and verifySignature() verifies the RSA signature of the given input. Similarly, we assume the existence of certain variables such as time, localBlockchain, and other self-explanatory values.

Sending a transaction: Voting

Algorithm 18 describes the process an elector must follow to craft and cast a vote. The transaction is signed with the elector's private key, ensuring its authenticity. Once the transaction is created, it is broadcasted to the network and added to the pool of unprocessed transactions. The transaction will remain in the pool until it is validated by a validator. If the transaction is

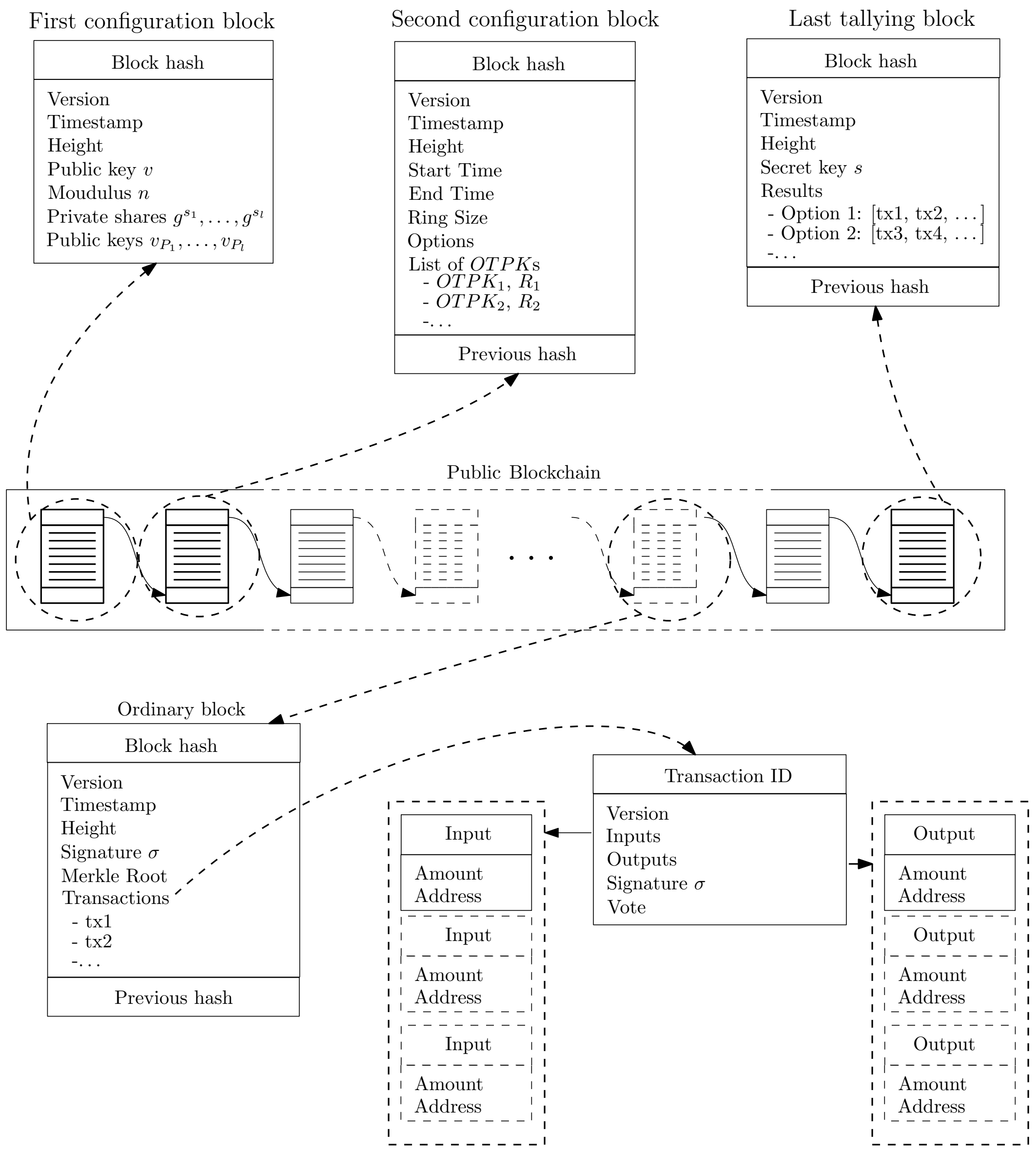


Figure C.1: Holistic view of all the data structures involved in the blockchain. We can appreciate the 4 different block types and its inner architecture.

deemed valid, it will be added to a new block on the blockchain. Otherwise, it will be removed from the pool, and the elector will need to create a new transaction to cast their vote.

Algorithm 18 Voting process(Craft vote and send transaction)

```
Require: voteDirection ← Vote direction.
    partyPublicKey ← Public key of the authority designated to add the block to the blockchain.
    publicKeyList ← List of public keys to form the Ring Signature.
    pivateKey ← Private key of the user.
    hiddenIndex ← Index in the list publicKeyList where the public key of the signer is.
    rsaPublicKey ← RSA public key to encrypt the vote direction until the tally.
    tokens ← Inputs containing he required tokens.
 1: mask ← randomMask()
 2: maskedVoteDirection ← (voteDirection || mask)
 3: encryptedVoteDirection ← $(maskedVoteDirection)^{rsaPublicKey}$
 4: ringSignature ← sign(publicKeyList.length, publicKeyList, hiddenIndex,
    privateKey, encryptedVoteDirection) (see Algorithm 6)
 5: broadcast(Transaction(
        Version: version,
        Inputs: tokens,
        Outputs: [partyPublicKey],
        Signature: ringSignature,
        Vote: encryptedVoteDirection,
        TransactionID: hash(Version, Inputs, Outputs, Signature, Vote)))
```

Handling transactions

To ensure the integrity of the voting process, parties must actively monitor the pool of unprocessed transactions for any new transactions addressed to them. These new transactions must be validated before they can be added to the blockchain. The process of validating new transactions is described in Algorithm 19.

The first step in the validation process involves checking whether the public keys used in the ring signature are included in the census of authorized electors. This step is crucial to ensure that only authorized electors are able to cast their votes. Once this is confirmed, the validity of the ring signature must be verified to ensure that the transaction has not been tampered with.

The final step in the validation process involves checking whether the inputs have enough tokens to operate on the blockchain. This ensures that parties cannot cast more votes than they are authorized to. If all of these validation steps are successful, the transaction can be added to the blockchain.

Algorithm 19 Validate Transaction

```
Require: transaction $\leftarrow$ Transaction to validate.
 1: if transaction.ID $\neq$ hash(transaction) then
 2:     return False
 3: end if
 4: if $\exists$ pk $\in$ transaction.Signature.PublicKeys | pk $\notin$ census then
 5:     return False
 6: end if
 7: if $\neg$ verifyRingSignature(transaction) then
 8:     return False
 9: end if
10: if $\sum_{i \in transaction.Inputs}$ < requiredTokens then
11:     return False
12: end if
13: validatedTransactions.append(transaction)
14: return True
```

Once nodes have received and validated a sufficient number of transactions, they can proceed to generate a new block and broadcast it to the rest of the network. The process of block generation involves selecting the transactions to be included in the block, ordering them, and adding a header to the block containing information about the previous block in the chain. Algorithm 20 outlines the steps required for generating a new block.

Handling blocks

Blockchain nodes must deal with the blocks received from other parties, verifying that they follow the longest chain and validating both the received block and its transactions. Algorithm 21 describes the process that nodes follow to validate the received blocks. If an invalid transaction is detected

Algorithm 20 Generating new blocks

```
Require: listTransactions ← List of validated transactions.
Require: version ← Version of the used protocol.
Require: lastBlock ← Previous last block added to the longest chain.
Require: localBlockChain ← Blockchain in the node's memory.
 1: version ← version
 2: timestamp ← time.now()
 3: previousHash ← lastBlock.ID
 4: height ← lastBlock.height + 1
 5: transactions ← listTransactions
 6: merkleRoot ← merkleTreeRoot(transactions)
 7: ID ← hash(version, timestamp, previousHash, height, transactions,
    merkleRoot)
 8: signature ← (version, timestamp, previousHash, height, transactions,
    merkleRoot, ID)$^{s_P}$
 9: newBlock ← Block(version, timestamp, previousHash, height, transac-
    tions, merkleRoot, ID, signature)
10: localBlockChain.append(newBlock)
11: $\forall_p \in$ Parties send($p$, newBlock)
```

during the process, as indicated in Line 9, the verifying node alerts the other nodes.

Algorithm 22 outlines the process for adding a block, deemed valid, to the longest chain. This algorithm presents a simplified logical representation of how nodes should follow the longest chain, and for an actual implementation, we refer the reader to the open-source code of Bitcoin. The algorithm assumes that the node maintains multiple chains in buffers, as Lines 10 and 12 exemplify. the node should employ a set of buffers for storing multiple chains. If the new chain includes blocks that are missing from the node's chain, the node must request those missing blocks from other nodes in the network, as shown in Line 14.

The interaction diagram shown in Figure C.2 provides an illustration of the process for casting and processing a vote. The diagram references the algorithms described above to depict the various stages of the election and the interactions between the parties and the blockchain.

While the diagram may simplify some of the more complex interactions between the parties and the blockchain, it provides an accurate representation

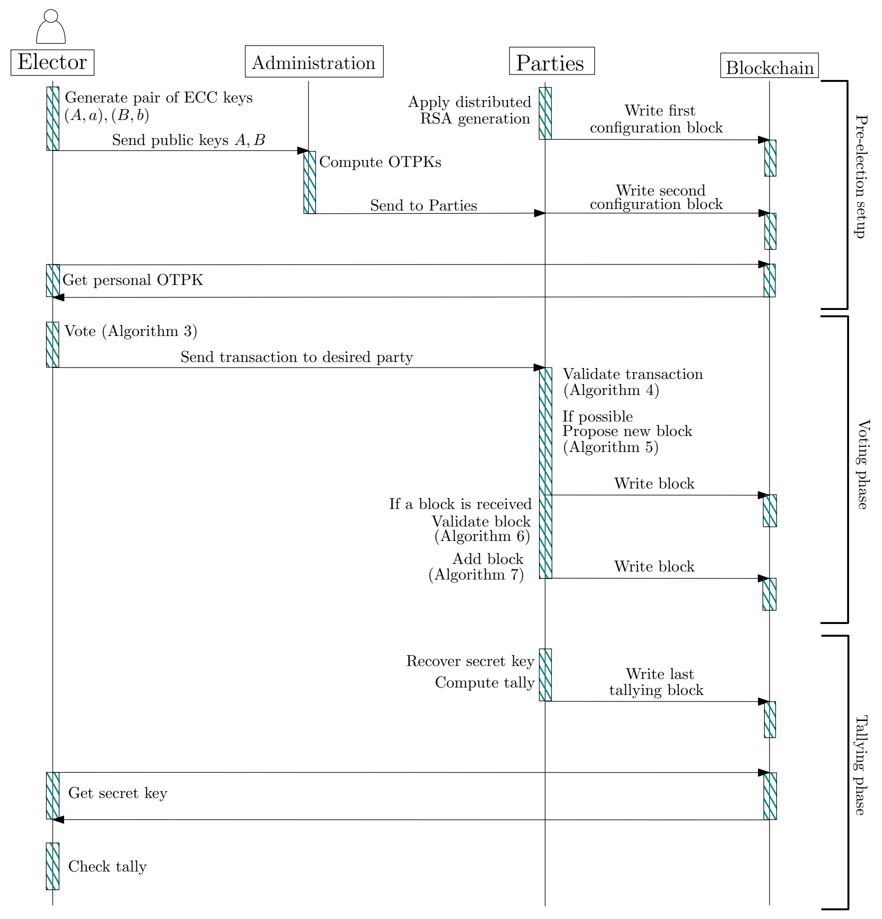
Algorithm 21 Validate block

```
Require: block ← Block to validate.
 1: if $\neg$ verifySignature(block.signature) then
 2:     return False
 3: end if
 4: if merkleRoot(block.transactions) $\neq$ block.merkleRoot then
 5:     return False
 6: end if
 7: for t $\in$ block.Transactions do
 8:     if t.ID $\notin$ validatedTransactions $\vee$
        $\neg$ validateTransaction(t) then
 9:         AlertError(block.ID, t.ID)
10:         return False
11:     end if
12: end for
13: return True
```

of the multiple stages of the election and the various partners involved in the process. By following the sequence of events depicted in the diagram, it is possible to understand how the voting process works and how the blockchain is used to ensure the security and transparency of the election.

Algorithm 22 Add Block

```
Require: block ← Block received through the network.
Require: localBlockChain ← Longest blockchain in the node's memory.
 1: if ¬ validateBlock(block) then
 2:     return False
 3: end if
 4: last ← localBlockChain.lastestBlock
 5: if last.ID = block.previousHash then
 6:     localBlockChain.append(block)
 7: else
 8:     if block ∉ localBlockChain then
 9:         if block.height < last.height then
10:             addToSecondaryChain(block)
11:         else
12:             saveAsSecondaryChain(localBlokchain)
13:             localBlockchain.append(block)
14:             ∀ b | b.height > last.height ∧
                b.height < block.height askForBlock(b)
15:         end if
16:     end if
17: end if
```


Elector
Administration
Parties
Blockchain
Generate pair of ECC keys
$(A, a), (B, b)$
Send public keys A, B
Apply distributed RSA generation
Write first configuration block
Compute OTPKs
Send to Parties
Write second configuration block
Get personal OTPK
Pre-election setup
Vote (Algorithm 3)
Send transaction to desired party
Validate transaction (Algorithm 4)
If possible Propose new block (Algorithm 5)
Write block
If a block is received Validate block (Algorithm 6)
Add block (Algorithm 7)
Write block
Voting phase
Recover secret key
Compute tally
Write last tallying block
Get secret key
Check tally
Tallying phase

Appendix D

How to Grant Anonymous Access Implementation

In this appendix, we present an implementation, and a empirical study on the time complexity of our identification protocols as described in Chapter 5. Our analysis demonstrates that for all three protocols, the computational complexity is predominantly $\mathcal{O}(log^3p)$, where p denotes the modulus. Additionally, we demonstrate that the computational complexity scales linearly with the number of users. However, in a real-world scenario, other factors as network latency may influence the presented results.

We implemented a Proof of Concept (PoC) of our three protocols to demonstrate their feasibility and complement our time complexity analysis. Empirical data illustrates aspects that cannot be considered in the theoretical complexity analysis, particularly those related to coordination and communication issues. We used JavaScript and Node.js to implement our protocols due to the distributed and asynchronous nature of the processes involved. We utilized the communication layer provided by the ZeroMQ[1] framework to facilitate communication. Please note that the PoC implementation includes some simplifications, such as the use of a centralized proxy to emulate a similar behavior to a real deployed implementation. The source code is available on Github[2], but it has not been properly audited and should not be used in production.

The PoC has been designed such that, in the TRA2 and TDRA2 pro-

[1]https://zeromq.org/

[2]https://github.com/Fantoni0/ara2

tocols, an increase in the number of guards and/or dealers used in the experiments results in a proportional increase in the number of terms in the polynomials utilized. For

We have implemented the PoC in a way such that, in the experimentation of TRA2 and TDRA2, the higher the number of guards and/or dealers, the higher the number of terms in the polynomials involved. Furthermore, this approach enables us to assess the impact of polynomial size on the overall system performance. However, it is important to note that the number of terms in the polynomial, and the number of parties involved may not necessarily be directly correlated.

The experiments were conducted using the Ubuntu OS and executed on a computer equipped with an AMD Ryzen 7 3700X CPU featuring 16 cores and 32 GB of RAM. To ensure accurate and reliable results, each configuration was executed 100 times and the resulting data was subsequently averaged to obtain statistically representative information.

The experimentation results have been summarized in three Figures: Figure D.1, Figure D.2, and Figure D.3. These Figures depict the total time required for the user to obtain access, segmented into two components: the time required for user registration, and the time required for guards to grant access to the resource.

It is worth highlighting that the reported performance is significantly impacted by the polynomial size. For instance, a scenario that employs 5 dealers and 8 guards (5D/8G) in the TDRA2 protocol requires the utilization of polynomials that contain five times more terms than a scenario that employs 1 dealer and 8 guards (1D/8G) in the TRA2 protocol. It is important to note, however, that the protocol can be implemented with smaller polynomials without compromising the security of the system. Additionally, it is pertinent to acknowledge that the number of guards employed in the protocol is directly proportional to the number of messages required to be exchanged to enable access, resulting in an increase in the total time required for access. Moreover, the number of terms in the polynomial utilized also plays a significant role in the system's overall performance, as demonstrated by experiments conducted with different configurations of dealers/guards employing polynomials of the same size (data not shown).

Regarding ARA2, we observed that the protocol provides acceptable delays while simultaneously offering anonymous access. However, it is important to note that ARA2 requires additional time to address synchronization issues arising from non-deterministic message ordering between the author-

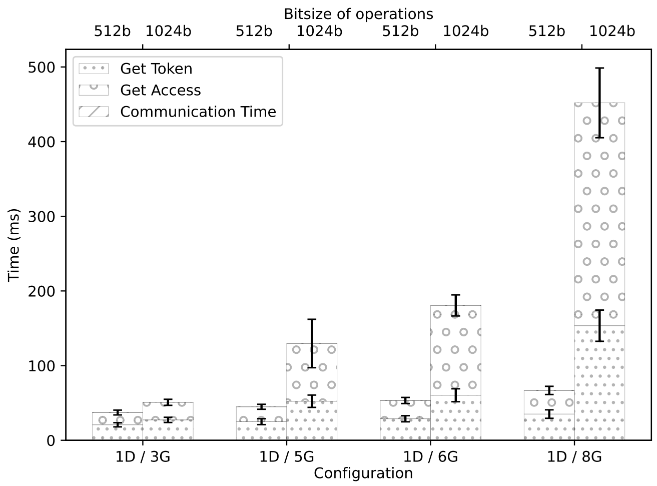


Figure D.1: TRA2 protocol. Experimental time results considering different configurations and bitsize of the operations.

ities, which is also present in other protocols albeit with a lower impact on the overall time.

We would like to note that the code utilized in this Appendix has been made open-source, enabling interested readers to run additional simulations and enhance the presented results. It is important to mention that the implementation has been tested with 512, 1024, and 2048 bit keys, but larger key sizes have not yet been tested. We welcome future improvements to the code through pull requests on Github. Some of the possible extensions that can be explored in the future include:

1. Incorporating checks to ensure that the protocol setup completes appropriately.

2. Updating the communication patterns to enforce the use of separate sockets for each party.

3. Replacing ZMQ sockets with websockets.

4. Adding encryption to communications between parties.

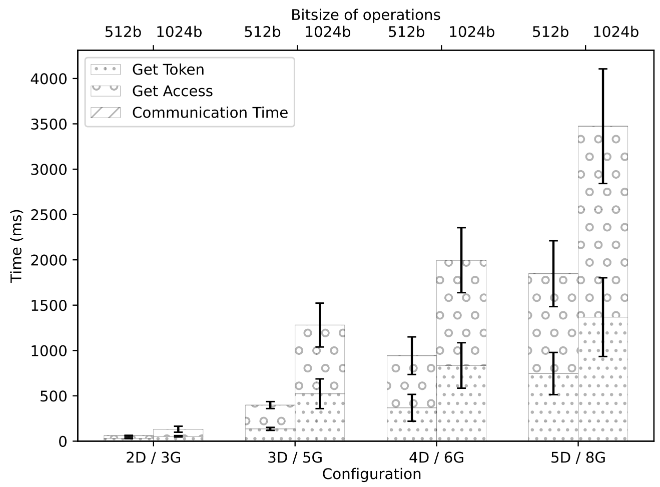


Figure D.2: TDRA2 protocol. Experimental time results considering different configurations and bitsize of the operations.

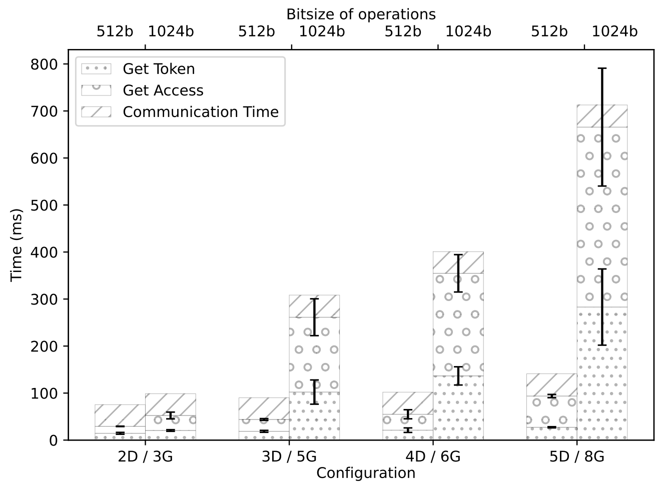


Figure D.3: ARA2 protocol. Experimental time results considering different configurations and bitsize of the operations.