Study of consensus protocols for blockchain technologies and quantum cryptanalysis of Misty schemes

Ambre Toulemonde

université
PARIS-SACLAY

Study of consensus protocols for blockchain technologies and quantum cryptanalysis of Misty schemes

Étude de protocoles de consensus pour les technologies blockchain et cryptanalyse quantique des schémas de Misty

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Mathématiques et Informatique
Graduate School : Informatique et sciences du numérique, Référent : Université de Versailles-Saint-Quentin-en-Yvelines

Thèse préparée dans l'unité de recherche LMV (Université Paris-Saclay, UVSQ, CNRS), sous la direction de **Jacques PATARIN**, Professeur, la co-supervision d'**Aline GOUGET**, Directrice en cryptographie avancée.

Thèse soutenue à Versailles, le 23 juin 2022, par

Ambre TOULEMONDE

Composition du jury

Louis Goubin Professeur, Université de Versailles Saint-Quentin-en-Yvelines	Président
Sébastien Canard Ingénieur de recherche, Orange Innovation	Rapporteur & Examinateur
David Naccache Professeur, Ecole normale supérieure (DI/ENS)	Rapporteur & Examinateur
Jean-Sébastien Coron Professeur, Université du Luxembourg	Examinateur
Jean-Paul Delahaye Professeur émérite, Chercheur au CNRS/Centrale Lille Institut/Université de Lille	Examinateur
Aline GOUGET Directrice en cryptographie avancée	Examinatrice
Jacques PATARIN Professeur, Université de Versailles et Thales DIS	Directeur de thèse

THESE DE DOCTORAT

NNT : 2022UPASG041

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Étude de protocoles de consensus pour les technologies blockchain et cryptanalyse quantique des schémas de Misty

Mots clés : Consensus, blockchain, élection de leader, cryptanalyse quantique, Misty

Résumé : Ces dernières années, deux domaines de recherche en cryptologie ont reçu une attention considérable : les protocoles de consensus pour les technologies blockchain dus à l'émergence des cryptomonnaies, et la cryptanalyse quantique due à la menace des ordinateurs quantiques. Naturellement, nos sujets de recherche se sont orientés vers ces deux domaines que nous avons étudiés séparément dans cette thèse. Dans la première partie, nous analysons la sécurité de certains protocoles de consensus pour les technologies blockchain qui sont un des principaux défis dans ces technologies. En particulier, nous nous focalisons sur l'élection de leader qui intervient au sein d'un protocole de consensus. Après une étude de l'état de l'art des protocoles de consensus avant et après l'introduction des technologies blockchain, nous étudions la sécurité de deux approches prometteuses pour construire des protocoles de consensus, appelées Algorand et Single Secret Leader Election. Ainsi, nous définissons un modèle de sécurité d'élection de leader avec cinq propriétés de sécurité qui empêchent les problèmes et attaques bien connus des protocoles de consensus. Ensuite, nous décrivons un nouveau protocole d'élection de leader appelé LEP-TSP destiné à être utilisé dans des systèmes privés et prouvons que LEP-TSP satisfait les propriétés de sécurité attendues tant que plus de deux tiers des participants sont honnêtes. Comme travail additionnel, nous présentons une description à haut niveau d'un nouveau protocole de consensus appelé Useful Work, qui utilise la puissance de calcul pour résoudre n'importe quel problème concret. Dans la seconde partie de cette thèse, nous passons en revue les meilleurs résultats de cryptanalyse connus sur les schémas de Misty et nous présentons de nouveaux résultats de cryptanalyse quantique. Tout d'abord, nous décrivons des attaques quantiques non adaptatives à clairs choisis (QCPA) contre les schémas de Misty L à 4 tours, Misty LKF à 4 tours, Misty R à 3 tours et Misty RKF 3 à tours. Nous étendons l'attaque QCPA contre les schémas de Misty RKF à 3 tours pour obtenir une attaque quantique permettant de retrouver la clé secrète des schémas de Misty RKF à d tours. Comme travail supplémentaire, nous montrons que la meilleure attaque non quantique connue contre les schémas de Misty R à 3 tours est optimale.

Title : Study of consensus protocols for blockchain technologies and quantum cryptanalysis of Misty schemes

Keywords : Consensus, blockchain, leader election, quantum cryptanalysis, Misty

Abstract : In recent years, two research domains in cryptography have received considerable attention : consensus protocols for blockchain technologies due to the emergence of cryptocurrencies, and quantum cryptanalysis due to the threat of quantum computers. Naturally, our research topics are geared towards these two research domains that are studied separately in this thesis. In the first part, we analyze the security of consensus protocols which are one of main challenges in these technologies. We focus more specifically on the leader election of consensus protocols. After a study of the state of the art on consensus protocols before and after the emergence of blockchain technologies, we study the security of two promising approaches to construct these protocols, called Algorand and Single Secret Leader Election. As a result, we define a security model of leader election with five security properties that address well-known issues and attacks against consensus protocols. Then, we provide a new leader election protocol called LEP-TSP intended to be used in private setting and prove that LEP-TSP meets the expected security properties while more than two third of participants are honest. As additional work, we provide a high level description of a new consensus protocol called Useful Work that uses the computing power to solve any real world problem. In the second part of this thesis, we review the best known cryptanalysis results on Misty schemes and we provide new quantum cryptanalysis results. First, we describe non-adaptive quantum chosen plaintext attacks (QCPA) against 4-round Misty L, 4-round Misty LKF, 3-round Misty R and 3-round Misty RKF schemes. We extend the QCPA attack against 3-round Misty RKF schemes to recover the keys of d-round Misty RKF schemes. As additional work, we show that the best known non-quantum attack against 3-round Misty R schemes is optimal.

Remerciements

Je souhaite tout d'abord remercier Aline Gouget et Jacques Patarin qui m'ont encadrée tout au long de cette thèse. Je leur adresse toute ma gratitude pour le temps et la confiance qu'ils m'ont accordés, ainsi que pour tous leurs conseils sur les différents sujets que nous avons pu développer ensemble.

J'adresse mes remerciements à Sébastien Canard et David Naccache d'avoir accepté d'être rapporteurs de cette thèse. Je les remercie d'avoir consacré du temps à cette lecture et pour m'avoir fait part de leurs remarques qui m'ont permis d'améliorer ce manuscrit. Je tiens également à remercier Jean-Sébastien Coron, Jean-Paul Delahaye et Louis Goubin pour avoir accepté de participer à mon jury de thèse.

Je remercie l'équipe de Thales qui m'a accueillie pendant cette thèse. Un grand merci pour tous nos échanges, professionnels ainsi que personnels, et tous les bons moments passés dans l'équipe qui m'ont permi d'avoir un cadre exceptionnel pour mes recherches. Je tiens à remercier particulièrement Amira pour toutes nos discussions et ses conseils qui m'ont aidée tout au long de cette thèse.

Je remercie également l'équipe de Smart Chain de m'avoir accueillie ces derniers mois. J'ai pu approfondir mes connaissances dans le monde de la "crypto" et finir sereinement l'écriture du manuscrit grâce à vous.

Merci également à mes amis du master CSI de Bordeaux avec qui j'ai partagé mes premiers pas dans la cryptographie et également ceux à Paris. Rémi, Thibault, Léa, Anthony et Quentin, je vous souhaite que le meilleur pour la suite.

Je tiens à remercier mes amis qui sont comme ma seconde famille : Rob, Q2, Jesse, Noemia, Habibi, Nini, Ben, Boti, Maxime, Tanguy, Thomas, Yoann, Delphine, Thib et Megan. Merci pour tous les moments partagés avec vous, principalement animés de fous rire et d'ivresse.

A mes plus vieux amis, David, Agathe et Mathilde, qui m'encouragent et me soutiennent depuis des années. Merci d'avoir toujours eu les bons mots, notamment pendant ces trois dernières années, qui m'ont aidé à balayer mes doutes et rempli de fierté.

Mes derniers remerciements vont évidemment à ma famille pour leur soutien. Merci d'avoir cru en moi et encouragé dans tout ce que j'ai entrepris. Je tiens à remercier en particulier ma soeur Aude pour son soutien et son aide précieuse pendant cette thèse.

Contents

General introduction

In a world where everything tends to be connected, the security of information and communications between each device needs to be guaranteed. More precisely, it is necessary to ensure confidentiality and integrity of data exchanged between individuals, businesses and governments. Cryptology which is the science of designing secure communication has become essential to address this security goal.

Cryptology encompasses two domains: *cryptography* that is the study of defense techniques and *cryptanalysis* that is the study of attack techniques. There exists two approaches in cryptography. First, secret-key or symmetric cryptography where the secret is shared between the sender and the receiver to encrypt and decrypt data. Second, public-key or asymmetric cryptography which uses a pair of keys linked mathematically, one public key is used by the sender to encrypt data and the other one is stored privately by the receiver to decrypt. Thus, cryptography enables to design encryption, signature schemes and authentication protocols that are the core of secure communication systems.

In recent years, two research domains in cryptology have aroused great interest: *cryptography in blockchain technologies* due to the emergence of cryptocurrencies, and *quantum cryptanalysis* due to the threat of quantum computers. Naturally, our research topics are geared toward these two areas. The manuscript addresses separately these two subjects.

Cryptography in blockchain technologies. *Blockchain technologies* became well-known in 2008 with the release of Bitcoin paper [85]. A blockchain is a digital ledger or a chain of blocks implemented in distributed manner, *i.e.* without using a central authority to maintain the ledger. Moreover, the blockchain is designed to be immutable. In other words, once a block of data is written into the blockchain, it cannot be removed or changed. In [85], the author (or the authors) known as Satoshi Nakamoto presents the *Bitcoin cryptocurrency* system. This is a digital money system in which transactions between users are grouped into blocks and publicly stored in the blockchain.

Even if blockchain technologies have been firstly used to develop cryptocurrencies such as Bitcoin and Ethereum, the first two cryptocurrencies created and listed on

market capitalization websites, these technologies proved to be relevant for several other applications. Indeed, transfers of cryptocurrencies generalized to transfers of digital information can be applied in a variety of sectors. For example, blockchain technologies can be used in supply chains to improve the traceability of products or in energy markets to facilitate energy transactions between prosumers who are simultaneously producers and consumers. Compared to current approaches, blockchain technologies can improve, facilitate and accelerate processes in various sectors.

Rapidly, two types of blockchain have been identified: public and private. In a public blockchain such as the one in the Bitcoin system, anyone at any time can read and add new blocks into the ledger. However, the public aspect is not necessary or even not desired in some applications. Thus, the private blockchain restricts only a group of authenticated participants to read and participate in the protocol. Depending on the use case, the type of blockchain needs to be suitably chosen.

Whatever the type of blockchain, one main process of these technologies is the storage of transactions that has to be done in secure manner, *i.e.* data stored into the ledger have not been tampered and cannot be modified or deleted. This process is executed by participants thanks to a *consensus protocol* that uses several cryptographic techniques that prevent from altering the ledger. A consensus protocol is a process that enables to reach a common agreement in distributed manner. In a blockchain context, participants use a consensus protocol to agree on which block of transactions is added into the ledger. Generally, these consensus protocols are based on leader election process that chooses randomly and in distributed manner one of the participants as leader who wins the right to write the next block into the ledger.

For example, in the consensus protocol used in Bitcoin, called *Proof-of-Work* (PoW), the leader is the first participant who solves a cryptographic puzzle known as *hash puzzle*. Participants have to find a block whose hash value is lower than a target value. However, several studies outline important issues in the Bitcoin PoW protocol such as the energy waste, the centralization of participants with the most computing power and many others.

Actually, designing consensus protocols is not a new problem and has been widely studied in distributed systems. With the new requirements of open participation and issues related to the Bitcoin PoW protocol, works to design new consensus protocols for blockchain while leveraging the first results for distributed systems are actively being researched.

In the light of the increased popularity of blockchain, the security of these technologies has to be studied. Several attacks, strategies, formalization and formal proofs have been released. In this thesis, we study the security of consensus protocols

that are one of the main challenges in blockchain technologies. In particular, we want to understand what is the level of trust and what are the security parameters of consensus protocols for blockchain.

Quantum cryptanalysis. Quantum computers are machines that exploit quantum physic instead of standard electronics enabling thus to perform tasks faster than classical computers. Roughly speaking, classical computers are based on binary computing where data are represented by bits that can take two values, either 0 or 1. Whereas in the quantum computers, data are represented by quantum bits, also called qubits that can be not only either 0 or 1 but also any superposition of 0 and 1.

The quantum cryptography is the science that exploits quantum mechanics to design new cryptography protocols such as quantum key distribution, and quantum cryptanalysis is the study of attack techniques using quantum computers. The term of post-quantum cryptography refers to protocols that uses classical computers and withstand attacks using classical and quantum computers.

In 1994, Peter Shor [96] outlined the threat of quantum computers that could break current systems based on asymmetric cryptography. Indeed, the security of these systems is based on problems assumed to be difficult to be solved with classical computers such as the computation of discrete logarithm or the factorization of large numbers. In theory, quantum computers could efficiently solve these problems and therefore break asymmetric cryptography systems yielding communication insecure. Regarding protocols based on symmetric cryptography, quantum computers could also be a threat but less important one. Indeed, in 1998, Grover [59] proposed a quantum algorithm allowing to make an exhaustive search among n elements in $O(\sqrt{n})$ time instead of $O(n)$. This can easily be mitigated by doubling the key size.

Even if today quantum computers that exist are not enough powerful to break current cryptography, the study of new quantum attacks and the design of post-quantum algorithms is important. Since 2017, the National Institute of Standards and Technologies (NIST) has launched a competition to standardize post-quantum public key algorithms. This competition aims to provide several standards for different applications, expected to be published within the next two to five years. The competition is still in progress at the time of writing.

In this thesis, we study quantum resistance of symmetric cryptography schemes called Misty. In particular, we provide quantum cryptanalysis of these schemes that have been used to design Kasumi algorithm, adopted as the standard cipher in the third generation mobile systems.

Contributions

This thesis addresses separately the research related to consensus protocols for blockchain technologies in a first part and the research related to quantum cryptanalysis in a second part.

Firstly, we present the work on consensus protocols for blockchain technologies that aim to provide security analysis of consensus protocols. We start by a state of the art on consensus protocols before and since the advent of blockchain technologies. Then, we present three contributions that analyze the security of consensus protocols. These works aim to outline security properties that address well-known issues and attacks. Next, we provide a fourth contribution which describes at high level a new consensus protocol that uses the computing power for useful works. We summarize these contributions in the following paragraphs.

1. Revisiting security analysis of Single Secret Leader Election. In this contribution, we revisit the security model of the Single Secret Leader Election (SSLE) [15]. We focus on the shuffling-based SSLE construction which is a practical scheme relying on the hardness of the Decisional Diffie-Hellman. SSLE schemes select exactly one leader whose identity remains hidden until he decides to reveal it. We first add the liveness property in the security model of SSLE schemes to ensure the election of a leader even in presence of malicious or inactive participants. Liveness is a classical security property in distributed systems that guarantees new data generated by leaders to be continually added into the system. Then, we refine the unpredictability property that means that non-leaders cannot guess who is the leader until he reveals his identity. We guarantee that there is at least two uncorrupted participants in the list from which the leader is elected and prevent the adversary learning which participant has been elected. We revisit also the fairness property that ensures that each participant has the same probability of being elected. Next, we provide a strategy enabling to introduce a bias so as to make one of corrupted participants elected with a probability of $\frac{f}{n}$ where f is the number of corrupted participants compared to a uniform probability of $\frac{1}{n}$. We propose a slight modification of the original scheme in order to achieve the fairness property. Finally, we motivate the need for our definition of the fairness property.

2. Unpredictability properties in Algorand. In this contribution, we analyze the unpredictability properties in the Algorand leader election protocol considered as a Probabilistic Leader Election. In Algorand, one or several potential leaders may be selected and a rule enables to choose one of them as leader. It is possible that there is no leader elected for some elections. Unpredictability originally defined in [15] is important to prevent an attacker from targeting the leader in DoS attack. First, we show that this unpredictability property is satisfied by Algorand. Then, we extend

this unpredictability to the t-forward unpredictability to capture the case where an adversary elected as leader cannot predict the leaders of the t following elections. This may prevent an adversary to plan which data to add at a suitable time into the ledger, *e.g.* transactions executed at a suitable time may be advantageous in high frequency trading. Finally, we describe a strategy for the expected number of potential leaders set to $n_l = 1$ enabling an adversary to be leader and predict the t next leaders with a probability of $\frac{f}{n}((\frac{f}{n})^t + (1-(\frac{f}{n})^t)(\frac{f}{n})^t) + (1-(\frac{f}{n})^t)\frac{1}{n-f} \cdot \frac{1}{n^t}$ instead of $(\frac{f}{n} + (1-\frac{f}{n})\frac{1}{n-f})\frac{1}{n^t}$ with at most f participants are corrupted among the n participants. In this contribution, we outline the properties related to the unpredictability and the importance of the parameter n_l of Algorand.

3. Security model and application to LEP-TSP. In this contribution, we define a security model of Single Leader Election (SLE) protocols with five security properties: uniqueness, fairness, unpredictability, t-forward unpredictability and liveness. They are defined in the model to address well-known issues and attacks targeting consensus protocols. Uniqueness means that exactly one leader is chosen in each election. The others properties have been studied in the previous contributions and are adapted in this work when it is necessary. Then, we propose a SLE construction called LEP-TSP which is a new leader election protocol based on external RNG services. It is intended to be used in private setting such as the private blockchain. We prove that our LEP-TSP protocol meets the expected security properties of a SLE protocol. In particular, LEP-TSP operates while $f < \frac{n}{3}$ participants are corrupted by an adversary, with n the total number of participants.

4. Useful work. As additional work, we propose a new consensus protocol called Useful Work (UW) where the computing work and the memory space are dedicated to useful works. Specifically, instead of solving a hash puzzle as in the Bitcoin protocol, participants run the code of any real world problems submitted by clients to be a candidate for winning useful coins. We also present some new issues and show that our UW protocol is resilient to these issues and the classical attacks on consensus protocols. This contribution aims to give an insight on what we can do with computing power instead of using it to solve the Bitcoin PoW puzzle. We present a high level description of UW protocol comprised of different mechanisms that may be done in several manners. This work can be used as basis for further studies to construct a consensus protocol that solves any* real world problems.

Then, in the second part dedicated to quantum cryptanalysis of Misty schemes, we present the following contribution.

*Assuming problems with a number of computations less than a computational complexity.

5. Quantum cryptanalysis of Misty schemes. We provide a quantum cryptanalysis of Misty schemes that aims to improve the results already known in the classical model. Misty schemes are symmetric schemes and well-known variants of Feistel construction used to design block cipher encryption protocols, here seen as pseudo-random permutation generators. We describe non-adaptive quantum chosen plaintext attacks against 4-round Misty L and Misty LKF schemes, and against 3-round Misty R and Misty RKF schemes. These attacks enable to distinguish these Misty schemes from random permutations in polynomial time. We extend the quantum distinguishing attack against 3-round Misty RKF schemes to obtain a quantum key recovery attack against d-round Misty RKF schemes with complexity $\tilde{\mathcal{O}}(2^{(d-3)n/2})$. Finally, we present a security proof with the same bound $2^{n/2}$ which shows that the best known cryptanalysis against Misty R schemes is optimal.

Organization

Part I is dedicated to the study of consensus protocols for blockchain. Chapter 1 gives the context of this first research topic. In Chapter 2, we provide the main notions used in this part. Chapter 3 and Chapter 4 provide a state of the art on consensus protocols. We give the security analysis of Single Secret Leader Election in Chapter 5 and the one of Algorand in Chapter 6. Chapter 7 provides the security model of leader election and the new protocol called LEP-TSP. In Chapter 8, we describe the new protocol of Useful Work. We summarize in Chapter 9.

Part II is dedicated to the study of Misty schemes. Chapter 10 gives the context of this second research topic. In Chapter 11, we provide the main definition used in this part. Chapter 12 gives an overview of previous works and the new results. In Chapter 13, we present quantum attacks against Misty schemes and the security proof of Misty R schemes with 3 rounds. We summarize this second part in Chapter 14.

Finally, we provide a general conclusion and give an extended abstract in French in last chapters.

Part I

Consensus protocols for blockchain technologies

Chapter 1

Introduction

Contents

In this first chapter, we give an overview of consensus protocols and new challenges to construct these protocols for blockchain technologies. We provide also some examples of blockchain applications to outline its popularity in several sectors. Finally, we present our contribution on this research topic and the organization of Part I.

1.1 Context

Consensus protocols are processes enabling a group of participants to reach a common agreement in distributed manner, *i.e.* without a central authority. These protocols are originally proposed as solutions to a problem introduced by Lamport *et al.* [75] referred as the *Byzantine generals problem*. In this problem, a group of Byzantine generals without a central entity attempts to agree on attacking or not an enemy city. The Byzantine generals problem became the seminal work for the research related to consensus protocols in distributed systems.

For decades, several works were undertaken to resolve this problem such as the *Practical Byzantine Fault Tolerant* (PBFT) [24] protocol introduced by Castro and Liskov. The PBFT protocol became the reference to achieve the basic security properties of *safety* and *liveness* in *partial synchrony* by assuming at most $\frac{1}{3}$ of the group may be corrupted. The safety property means that (i) only a value that has been proposed may be chosen; (ii) only a single value is chosen, and (iii) a participant never considers that a value has been chosen unless it actually has

been. The liveness property requires that a consensus can be achieved even if some fraction of participants may be malicious or inactive. The partial synchrony assumption means that messages can be delayed, duplicated or delivered out of order.

With the advent of the blockchain technology proposed by Nakamoto [85], an increased interest in consensus protocols has emerged. Participants use a consensus protocol to select randomly and in distributed manner one of them as leader who provides the next block of data to be added into the ledger. This process is also known as *leader election*.

The well-known *Proof-of-Work* (PoW) [85] consensus protocol has been introduced to select the leader proportionally to his computing power. Moreover, the Nakamoto PoW protocol is designed to achieve the new requirements of *scalability* and *incentivation* of a public blockchain. The scalability enables to handle a variable and huge number of participants, and the incentivation aims to motivate entities to participate in the consensus protocol. However, the Nakamoto PoW protocol has a number of limitations such as an intensive energy consumption, the fork problem, the selfish mining strategies, etc.

Thus, one of the main challenges of blockchain technologies is the choice of consensus protocols. The security of these technologies mainly relies on the security provided by consensus protocols*. Hence, with the new requirements of blockchain technologies and issues of the Nakamoto PoW protocol, new consensus protocols need to be designed while leveraging the advantage of first results to solve the Byzantine generals problem. Analyzing the security of consensus protocols is important to evaluate the trust that we can have in blockchain technologies. Therefore, this thesis provides security analysis of consensus protocols by focusing on the leader election process.

Blockchain applications. Blockchain technologies became a promising technology due to its new way of trust. Indeed, it allows participants who have more or less trust in each other to work together, *i.e.* exchanging information in secure manner and without the need of a central authority. Its popularity begun with the Bitcoin cryptocurrency [85], a fully distributed digital money system based on a blockchain technology. Indeed, previous works as Digicash [27], B-money [34] or Bit Gold [99] were first solutions of digital money systems but required a trusted party or are vulnerable to Sybil attack where several identities are created by an attacker to increase her influence in the protocol.

Other companies have created their own cryptocurrency based on blockchain technologies by improving the idea of Nakamoto such as Ethereum [21], Cardano [63]

*Other factors may impact the security of blockchain technologies, such as the used programming language or the implementation of algorithms, that are out of scope of our work.

and Algorand [29]. Even if blockchain technologies have been mainly deployed for financial transactions, it is increasingly used in other domains such as in the supply chain [82, 42], medical sector [92, 71] or energy markets [4, 67].

The supply chain is responsible for the management of the flow of millions of products, and the *traceability* is one of its most important aspect. Traceability enables to identify and trace the product pieces, and to know the path they took to their final destinations. The blockchain technology can improve the traceability aspect of supply chains by guaranteeing different properties: trade-privacy of who and what are involved in a shipping of product and transparency of the immutable amount of products [82], or the privacy-sensitive information, certificate verifiability and auditability [42].

The blockchain may also facilitate the exchanges in health care sector for the entry and operation of clinical data without compromising other sensitive data [92] or the management of medical supplies [71].

Another example of a blockchain use case is given in [4, 67] regarding the energy sector. The blockchain may be designed as an energy market to facilitate and help energy transactions between *prosumers* who are simultaneously producers and consumers. All information related to the energy flows and services can be stored into the ledger. Thus, this may ensure energy provenance, transactions privacy and immutability of data.

1.2 Contribution and Organization

Constructing or selecting a consensus protocol is one of main challenges in distributed systems, and for the last couple of years, in particular for the blockchain technologies due to their promising benefits in several domains. New consensus protocols for these technologies have been developed in order to achieve the safety and liveness properties while avoiding the issues of the Nakamoto PoW protocol. Several papers on the security analysis of these protocols that exhibit the issues of well-known Nakamoto PoW protocol have been published in the literature.

First, we present a state of the art on consensus protocols. Then, we study two promising approaches to construct consensus protocols named Single Secret Leader Election (SSLE) and Algorand. In particular, we focus on the leader election process which is an important mechanism in consensus protocols, and outline important security properties to prevent well-known issues. Next, we provide a new security model for leader election protocols with these security properties. We propose also a new leader election protocol, named LEP-TSP, intended to be used in private setting and prove that LEP-TSP meets the expected security properties. Finally, we provide a high level description of a new consensus protocol named Useful Work intended to make computation power useful.

This part is organized as follows. In Chapter 2, we provide the main notions and cryptographic primitives used in Part I. Then, we present a state of the art on consensus protocols before the emergence of blockchain technologies in Chapter 3 and for blockchain technologies in Chapter 4. Chapter 5 and Chapter 6 present the work on SSLE and Algorand respectively. In Chapter 7, we propose a security model of leader election protocol with five security properties that addresses well-known issues and attacks targeting consensus protocols. We describe also LEP-TSP protocol which is a new leader election protocol based on external RNG services, intended to be used in private setting. The last contribution described in Chapter 8 proposes a new consensus protocol called Useful Work. Finally, we summarize this part in Chapter 9.

Chapter 2

Cryptographic primitives and main notions

Contents

In this chapter, we provide the main notions and cryptographic primitives used in the first part of this manuscript.

2.1 Cryptographic primitives and assumptions

All along this part, we mention and use the following cryptographic primitives and assumption. Mostly, we consider that the primitives satisfy the desired properties, unless otherwise specified, and outline which property is the most significant according to the context when it is necessary.

2.1.1 Decisional Diffie-Hellman assumption

Let $\mathbb{G}$ be a finite group of prime order q, the Decisional Diffie-Hellman assumption [37] states that, given a generator $g \in \mathbb{G}$, two elements $g^a, g^b \in G$ and a candidate $X \in G$, it is hard to decide whether $X = g^{ab}$ or not.

2.1.2 Hash functions

Hash functions map arbitrarily long bit-strings to strings of fixed length. A hash function $HASH$ takes a message $x \in \{0,1\}^*$ as input and outputs a value $y \in \{0,1\}^n$ referred to as a *hash value* or simply *hash*. The hash function $HASH$ must guarantee the following security properties [1]:

- *Preimage resistance*: for most y in $\{0,1\}^n$, it is hard to find a string x such that $HASH(x) = y$.
- *2nd-preimage resistance*: given x, it is hard to find a string $x' \neq x$ such that $HASH(x) = HASH(x')$.
- *Collision resistance*: it is hard to find two different strings x and x' such that $HASH(x) = HASH(x')$.

In this manuscript, the term hard means that there is no polynomial-time algorithm that enables to break the security property.

2.1.3 Digital signature schemes

Digital signatures are schemes that ensure authenticity, non-repudiation and integrity of digital message. A digital signature scheme S is defined by three algorithms $(KGEN, SIGN, VERIF)$. The algorithm $KGEN$ takes as input a security parameter λ and generates the secret and public keys (sk, pk). The algorithm $SIGN$ takes as input a secret key sk and a message x, and outputs a signature *sign*. The algorithm $VERIF$ takes as input a public key pk, a message x and a signature *sign* and rejects or accepts the signature by outputting 0 or 1 respectively. A digital signature scheme must fulfill the following properties:

- *Unforgeability*: only the signer can produce a valid signature on a given message
- *Non-repudiation*: the signer cannot deny having signed the message with a valid signature
- *Integrity*: the content of message has not been modified

For the sake of simplicity, we use the terms of signature scheme or signature to designate a digital signature scheme.

2.1.4 Random number generator

A random number generator (RNG) is a function that provides a bit string r from a secret s and has the following properties:

- *Unpredictability*: anyone without the knowledge of s cannot produce better than a random guess on the value r prior to some step barrier in the generation of the value.
- *Randomness*: the value r is indistinguishable from a uniformly distributed random string for anyone without the knowledge of s.

2.2 Participants

We use the notion of participants to designate the ones who execute the protocol. We denote by n the total number of participants in a protocol. Thus, the set of n participants are denoted by $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$.

An *honest*, also called *uncorrupted*, participant designates an online participant who performs all the protocol instructions.

A participant is *malicious*, also called *Byzantine*, *corrupted* or *faulty*, if she deviates from the protocol instructions. We consider that the set of malicious participants may act as a single entity designated by an *attacker* or *adversary*.

Note that, offline participants may not be necessarily malicious [35] but they can seriously affect the protocol. We denote by f the number of malicious or offline participants and by h the number of honest participants. Thus, we have $n = f + h$. In this manuscript, we use *he/his* to designate an honest participant and *she/her* for a malicious participant.

2.3 Centralized, decentralized and distributed

Thanks to Figure 2.1, we can easily distinguish the different structures for a system. In a centralized model, there is a unique central entity that manages the other participants. Every modification and request require the agreement of this central party. Whereas, in a decentralized system, no unique party controls other. However, some participants depend on one or several other participants who may impact them. Finally, in a distributed model, all participants are equals and every modification needs a common agreement to be adopted. The protocols studied in this part are between decentralized and distributed organization according to the use case.

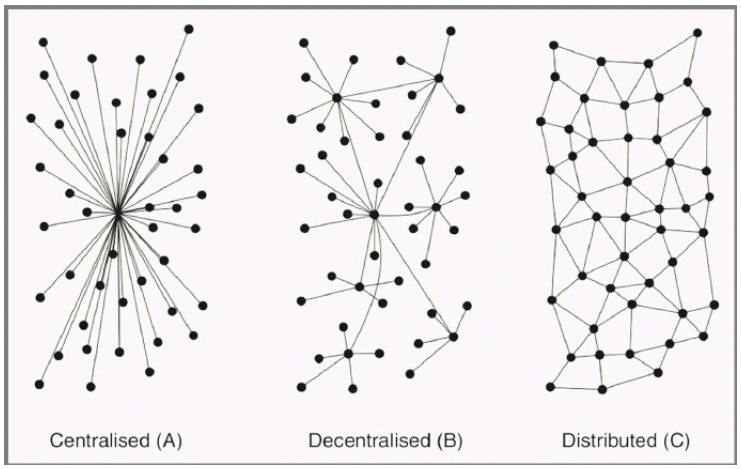


Figure 2.1: Three organizations for a system [61]

2.4 Consensus protocols

Consensus protocol is a central component of any distributed or decentralized system in which a group of n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ agree on the current state. Generally, a consensus protocol is divided in rounds where at each round k, an agreement on the state is reached. The basic properties of a consensus protocol are originally defined as follows:

- *Agreement*: all honest participant decides on the same value.
- *Validity*: if a participant decides on a value, then this value was proposed by some participant.
- *Termination*: every honest participant eventually decides on a value.

The basics security properties of distributed systems are *safety* and *liveness*. Firstly, they were informally introduced in [73] such that the safety property states that something bad will never happen and the liveness property means that something good will eventually happen. For consensus protocols, these properties are refined as follows [74]:

- *Safety*: (i) only a value that has been proposed may be chosen; (ii) only a single value is chosen, and (iii) a participant never considers that a value has been chosen unless it actually has been.
- *Liveness*: a consensus on a value can be achieved even if some fraction of participants may be malicious or inactive, *i.e.* offline or do not actively participate in the protocol.

2.5 Leader election protocols

In this thesis, a leader election protocol is considered as a particular case of consensus protocols. The n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ agree in distributed manner on a leader who may have a particular role. For example, in a context of distributed ledger, the leader provides the next data to add into the ledger. Generally, the leader election protocol is divided in rounds where at each round one or several leaders may be randomly elected. This is also possible that in some leader elections there is no leader for some rounds.

In this thesis, we study and specify the security properties of leader election needed to construct consensus protocols for blockchain based on the security properties defined in the previous section.

2.6 Blockchain technologies

Blockchain technologies are new technologies based on a data structure called *blockchain*. The blockchain structure is seen as a *ledger* or *chain of blocks* where each block stores transactions in distributed manner without a central authority. The blockchain structure is intended to be immutable, *i.e.* once a block of transactions is *written* or *added* into the ledger, it should no longer be removed or changed. A new block is added after reaching a *consensus* on this block. The blockchain structure can be considered as a distributed or decentralized system according to the use case.

In this manuscript, the notion of blockchain designate the blockchain structure where a new block of transactions is added after the participants reach a consensus on this block. The height of the blockchain corresponds to the number of blocks added after the first block called *genesis block*. This latter is generally configured by the initial participants. Figure 2.2 represents an ideal structure of blockchain of height k with the genesis block B_0. We designate also by *chain* or *branch*, all or a part of the blockchain. The *main* chain designates the chain chosen by participants generally in the case of a fork, *i.e.* several branches extending a same block.

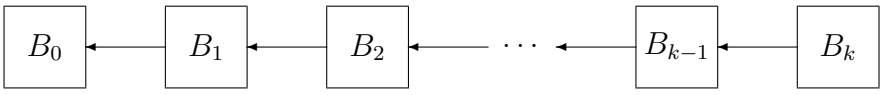


Figure 2.2: Blockchain of height k.

Actors. We can generally distinguish two groups of actors in a blockchain: *users* and *nodes*. Users can be persons, entities, organizations, businesses and others which transfer assets between each other. The role of users is to generate, sign and send *transactions*. A node is an individual system which is a part of the blockchain system. Each node may store a copy of the ledger. The nodes have the main role in the blockchain which is the responsibility to generate, verify and write transactions and so new blocks into the ledger. Obviously, a user may be also a node.

In this manuscript, we focus on the role of nodes which are the participants described in Section 2.2. We denote the set of nodes by $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$.

Transactions. A *transaction* may be a transfer of assets such as money, goods or information, *e.g.* the progress of a product in a supply chain. The transactions are generally generated, signed and sent by users. A transaction is considered as *valid* if the information related to this transaction are valid. For example, it contains valid signatures, the transaction issuer owns the assets, the address of transaction receiver exists, assets are not *double spent* and many others.

In this manuscript, the term of transactions designates any data added into the blockchain according to its use case.

Blocks. A *block* is a set of data usually created by a node. A block is typically divided into two parts, a *block header* and a *block data*. In the block data part, the node may include a list of valid transactions and the hash of the block header. Other data such as ledger events may be also included in this part. The block header part may include the information related to the block such as the block number, a timestamp, the Merkle tree root of the transactions list and others. The hash value of the previous block header is also included in the block header which enables to link the blocks with each others.

A Merkle tree [79] is a tree, usually binary, such that each node is the hash of its children. In a blockchain context, the hash of each valid transaction represents a leaf of a Merkle tree and the corresponding Merkle tree root is added into a block header.

Figure 2.3 gives an example block $B_{i,k} = (BHeader_{i,k}, BData_{i,k})$ generated by a node $\mathcal{P}_i$ with a Merkle tree with four valid transactions $tx0, tx1, tx2$ and $tx3$. The values $BHeader_{i,k}$ and $BData_{i,k}$ are the block header and block data respectively.

Writing a new block. The protocol to add a new block into the blockchain may be divided into these following phases.

Phase 1: `Issuing the transactions`. Users create transactions that transfer assets to other users and broadcast their transactions to nodes.

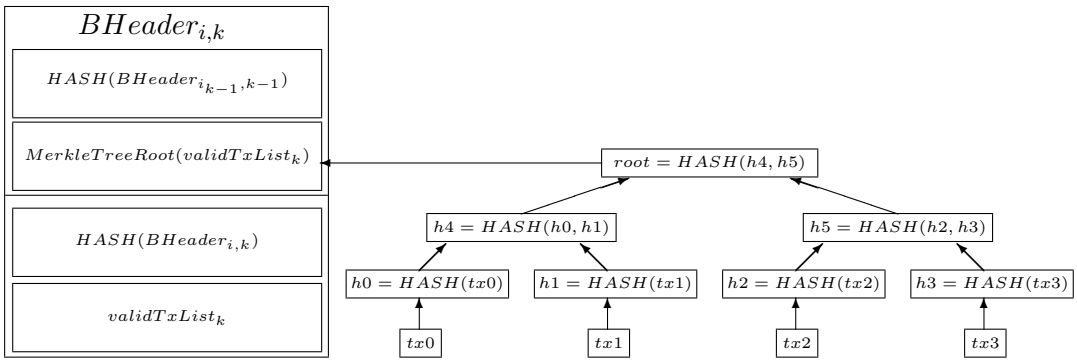


Figure 2.3: Example of block with the Merkle tree roof ot the list of valid transactions $validTxList_k = \{tx0, tx1, tx2, tx3\}$.

Phase 2: `Verification of transactions`. Nodes verify transactions and put valid ones on hold in a local pool of pending transactions*.

Phase 3: `Block construction`. Nodes create candidate blocks with valid transactions pending in their pool.

Phase 4: `Consensus`. Nodes use a consensus protocol to agree on the next block to be added into the ledger. An additional voting step can occur to consider the block as *confirmed*, *i.e.* it may be considered as an immutable part of the blockchain and so transactions included in it can be executed.

Obviously, these phases and their order may differ from a protocol to another. For example, Phase 4 may be a leader election that firstly elects a node as leader to execute Phase 3.

In this manuscript, we assume that Phase 1 and Phase 2 are already executed and we focus on Phase 3 and Phase 4. In other words, we consider that nodes have (different) pools of valid transactions and want to add the next block into the blockchain.

Blockchain categories. Currently, two models of blockchains may be considered: *permissionless* blockchain and *permissioned* blockchain.

A permissionless blockchain is also called a *public* or *fully open* blockchain. In this blockchain model, anyone can participate in the blockchain, *i.e.* being a user and/or a node. Moreover, all data written into the ledger such as messages and transactions between users and nodes are visible by anyone. A public blockchain relies also on a reward system to incentive nodes to maintain the blockchain. This requirement is usually called *incentivation*.

A permissioned blockchain is also known as a *private* or *consortium* blockchain. In this model, users and nodes know each others and want to keep a restricted access to the blockchain. Thus, only a defined group of participants, generally

*The pool of each node may be different from another due to problems of propagation or a malicious behavior.

also authenticated, can read and write data into the ledger. It is possible that this authenticated group restricts read access and who can issue transactions [102]. Furthermore, participants typically have a common interest, *e.g.* a business interest, in working together. There is then usually no real need to reward nodes to maintain the blockchain.

Table 2.1 summarizes criteria that differentiate the blockchain models and gives some examples of consensus protocols intended to be used in each blockchain model.

Categories	*Public blockchain*	*Private blockchain*
Generate and write the new block	Anyone when elected	A group
Read and verify the blocks	Anyone	A group
Authentication system	No anonymous nodes	Yes authenticated nodes
Data classification	Public	Restricted
Incentive	Yes, for writers	No
Consensus example	PoW, PoS	PBFT, PoET

Table 2.1: Characteristics for each blockchain model. PoW = Proof of Work, PoS = Proof of Stake, PBFT = Practical Byzantine Fault Tolerance and PoET = Proof of Elapsed Time.

Chapter 3

Consensus protocols before the advent of blockchain technologies

Contents

In this chapter, we present the original problems related to consensus protocols and the main resulting solutions.

3.1 Distributed agreement problems

In distributed agreement problems, a set of n participants, also called *nodes*, competes or cooperates to achieve the same goal which is *reaching a common agreement without a central authority*. There are three well-known and closely related distributed agreement problems [45]: the *interactive consistency problem* [90],

the *Byzantine generals problem* [75] and the *consensus problem* [46]. These problems differentiate from who provides the initial value(s) and on what is the agreement. Table 3.1 provides the starting values and final outcomes of these three problems.

Problem	*Interactive consistency*	*Byzantine generals*	*Consensus*
Who initiate the value(s)	All nodes	One node	All nodes
Final agreement	A vector of values	Single value	Single value

Table 3.1: The distributed agreement problems.

Each of these problems may address different practical applications. For example, the interactive consistency problem may be relevant for systems that rely on the combination of several opinions to provide a service [30]. The Byzantine generals problem can be applied in database management systems [51] where a user command is executed in each database stored by nodes and an agreed result has to be sent back to user, for example. Regarding the consensus problem, it may address the clock synchronization problem [45] where each node has an initial clock value and a periodic agreement on a single clock value is reached such that two honest nodes never differ by more than some value.

Common assumptions. Among the n nodes, up to f may be faulty. For these three problems, a faulty node means that the node can send altered messages or refuse to send messages. The $n - f$ other nodes are called non-faulty nodes. They communicate only by two-party messages, *i.e.* the sender and the receiver may alternate their role, and the receiver always knows the identity of the sender. Each node can communicate directly with any other node. For simplicity, we assume a binary agreement where the agreed values are in $\{0, 1\}$. Results can easily be extended to multivalue agreement [31]. We first detail the three problems and then we give the relation among them.

3.1.1 Interactive consistency problem

In the interactive consistency problem [90], each node has an initial value that may be different from others. Given $f, n \geq 0$, the non-faulty nodes try to compute a vector of values and to meet the following conditions:

- **Agreement**: all non-faulty nodes compute exactly the same vector.

- **Validity**: if the ith node is non-faulty and its initial value is v_i, then the ith value of the vector agreed by all non-faulty nodes must be v_i.

Note that if the jth node is faulty, then all non-faulty nodes can agree on any common value for v_j.

3.1.2 Byzantine generals problem

Relying on their work on the interactive consistency problem, Lamport *et al.* released the pioneer work on consensus protocols known as *the Byzantine generals problem* [75]. In the Byzantine generals problem, an arbitrarily chosen node, called *leader*, broadcasts his initial value to all other nodes. The non-faulty nodes try to reach an agreement on a single value that should satisfy the following conditions:

- **Agreement**: all non-faulty nodes agree on the same value.
- **Validity**: if the leader node is non-faulty, then every non-faulty nodes agree on the value sent by the leader

Note that, in the case where the leader is faulty, then all non-faulty nodes may agree on any common value.

3.1.3 Consensus problem

Another closely related problem which has been also studied extensively in the literature is the *consensus problem* [45, 46]. Every node has its own initial value that may be different from others. The goal of the consensus problem is to reach an agreement on a single value such that:

- **Agreement**: all non-faulty nodes agree on the same value.
- **Validity**: if the initial values of every non-faulty nodes is v, then all non-faulty nodes must agree on the value v

Note that, if the initial value of each non-faulty nodes are different, then all non-faulty nodes can agree on *any* value of a node.

3.1.4 Relations among the three problems

These three distributed agreement problems are closely related [45]. In particular, the Byzantine generals problem may be considered as a special case of the interactive consistency problem where the nodes is interested only in the initial value of one node. Thus, a protocol resolving the interactive consistency problem also solves the

Byzantine generals problem. Inversely, protocols resolving the Byzantine generals problem can be run in parallel for each node and enable to solve the interactive consistency problem. Moreover, a protocol that solves the interactive consistency problem can also be a solution for the consensus problem. Indeed, all non-faulty nodes can choose the majority value of the vector agreed by the protocol for the interactive consistency problem, or choosing a default value if the majority does not exist. Protocols solving consensus protocol needs an extra step to solve the two other problems as proposed in [45].

Therefore, protocols that solve the Byzantine generals problem can also be solutions for the interactive consistency problem and the consensus problem. These solutions are generally called consensus protocols. Thus, the Byzantine generals problem has been considered as the pioneer work to construct consensus protocols.

3.2 Possibility and impossibility results

In this section, we provide the first well-known possibility and impossibility results for the Byzantine generals problem and consensus problem.

3.2.1 Results under synchrony

First works have been studied under *synchrony* defined as follows: (a) the communication is synchronous, *i.e.* the message delivery time is fixed and known; (b) the execution speed is synchronous, *i.e.* nodes run at a fixed and known rate; and (c) each node has a clock and all clocks are synchronized. Thus, once a message is sent, it is also delivered correctly and an absence of message can be detected.

Three results for the Byzantine generals problem have been provided in the original paper [75] and recalled in this section:

1. It is not possible to solve the Byzantine generals problem with three nodes in presence of one faulty node.

2. It is possible to solve the Byzantine generals problem with $n \geq 3f + 1$.

3. Assuming that the messages are signed, it is possible to solve the Byzantine generals problem with any value of n and f*.

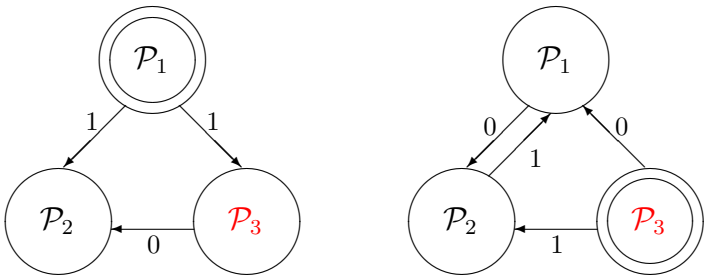
Impossibility when $n = 3$ and $f = 1$. The first result comes from the following scenario. We assume three nodes $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ such that $\mathcal{P}_3$ is faulty. The goal is to agree on the value 0 or 1, which refer to "retreat" or "attack" in the original

*The problem is considered as *empty* if $n \leq f + 2$ [75].

paper [75]. Two following cases are considered and represented in Figure 3.1: (1) non-faulty leader and (2) faulty leader.

In Case (1), the leader, for example $\mathcal{P}_1$, sends the value 1 to $\mathcal{P}_2$ and $\mathcal{P}_3$. The faulty node $\mathcal{P}_3$ reports to $\mathcal{P}_2$ that she has received the value 0. Thus, $\mathcal{P}_2$ has conflicting values but must choose the value 1 to satisfy Validity.

In Case (2), the leader $\mathcal{P}_3$ sends 0 to $\mathcal{P}_1$ and 1 to $\mathcal{P}_2$. Then, $\mathcal{P}_1$ communicates 0 to $\mathcal{P}_2$, and $\mathcal{P}_2$ communicates 1 to $\mathcal{P}_1$. Thus, $\mathcal{P}_2$ is constrained to choose 1 to satisfy Validity and $\mathcal{P}_1$ receiving 0 from the leader must choose 0 to satisfy Validity. Finally, $\mathcal{P}_1$ agree on 0 and $\mathcal{P}_2$ agree on 1, that violates the Agreement condition.



Case (1) Case (2)

Figure 3.1: Impossibility when $n = 3$ and $f = 1$. Double circle represents the leader and $\mathcal{P}_3$ is the faulty node written in red.

Solution when $n \geq 3f + 1$. Thus, based on the previous impossibility result, Lamport *et al.* proposed a solution referred to the *Oral Message* (OM) algorithm [75], that solves the Byzantine generals problem for $n \geq 3f + 1$ nodes where at most f nodes may be faulty.

OM algorithm works roughly as follows. First, the leader sends his value to every node. Next, each node acts as the new leader and sends the value received from the leader to the $n - 2$ other nodes. Then, the node sends the majority value received from other nodes. This step is performed f times.

Thus, in both cases of non-faulty and faulty leader, non-faulty nodes agree on the same value and both conditions of the Byzantine generals problem are satisfied. We refer the reader to the original paper [75] for the details of $OM(f)$ and its proof of correctness for any value of f.

Solution with signed messages for any n and f. The last result [75] is referred as the *Signed Message* (SM) algorithm [75] using unforgeable signature on messages to restrict the faulty node ability to alter messages. In particular, the

signature of non-faulty node cannot be forged, any alteration of the signed message content can be detected and anyone can verify the authenticity of a node signature. Compared to the case where messages are not signed as in OM algorithm [75], the nodes know that the leader is faulty since they receive two different values signed by the leader. We refer the reader to the original paper [75] for the details of $SM(f)$ and its proof of correctness for any value of f.

3.2.2 FLP impossibility in asynchrony

Another major work was undertaken by Fischer, Lynch and Paterson [46] known as the *FLP impossibility* that considers the consensus problem in *asynchrony* [46] defined as follows: (a) the communication is asynchronous, *i.e.* the message delivery time is not fixed and not known; (b) the execution speed is asynchronous, *i.e.* the speeds of nodes is not fixed and not known, and it is not possible to differentiate if the node has stopped entirely or is running very slowly; and (c) nodes have not access to a synchronized clock. The FLP impossibility is defined as follows:

> It is impossible to have a protocol that solves the consensus problem in asynchronous system in which only one node may crash,

where *crash* means that the node is non-faulty but may stop at any time and cannot restart.

3.2.3 Solutions under partial synchrony

Owing the FLP impossibility in asynchrony, the time assumptions have been relaxed in the following works on the Byzantine generals problem to obtain solutions intended to be used in practice. Indeed, even if the OM and SM algorithms [75] satisfy both conditions of the Byzantine generals problem, synchrony is a strong assumption that stops the protocol when synchrony is violated. These theoretical results do not enable to obtain robust results for practical applications where natural network problem may occur at any time, for example. Thus, *partial synchrony* originally introduced by Dwork *et al.* [40] may be a trade-off between synchrony and asynchrony assumptions. Partial synchrony may be defined as one of these following definitions [40]:

- (a) communication is partially synchronous, *i.e.* one of these two situations holds: the message delivery time exists but is not known or it is known and has to hold from some unknown point; and (b) the execution speed of nodes is synchronous
- (a) communication is synchronous; and (b) the execution speed of nodes is partially synchronous, *i.e.* the upper bound on the relative speeds of nodes

can exist but be unknown or it can be known but hold only from some unknown point

- communication and execution speed of nodes are partially synchronous

In their work [40], they consider these following four types of failures: *omission* that is a non-faulty node who fails to send or receive messages, *i.e.* some messages are lost in transit due to various causes such as transmitter malfunction or receiver out of range; *Byzantine* that is is a faulty node sending erroneous messages or not behaving as expected; *authenticated Byzantine* that is a Byzantine node who signs messages and his signature cannot be forged; and crash failure.

They provide results for the Byzantine generals problem with the additional following conditions:

- **Termination**: every non-faulty node eventually decide on a value

Based on [40], Table 3.2 compares algorithms [40] resolving the consensus protocol in partial synchrony with previous works [75, 46]. It presents the necessary number of nodes n where at most f are faulty to solve the consensus problem. The column Asynchronous contains ∞ for all lines due to the FLP impossibilty [46].

We can outline that a majority of non-faulty nodes is necessary in the two cases with a partially synchronous communication in presence of crash and omission failures. Moreover, the same result as in [75] for Byzantine fault is found in each case with partial synchrony assumptions which is $n \geq 3f + 1$ by considering at most f Byzantine faults. Lastly, in the two cases with a partially synchronous communication and Byzantine faults, the authentication does not improve the resiliency compare to the case with a synchronous assumption.

Failures	*Syn--chronous* [75]	*Asyn--chronous* [46]	*Partially synchronous communication and execution*	*Synchronous communication and partially synchronous execution*	*Partially synchronous communication and synchronous execution*
Crash	f^*	∞	$2f + 1$	f^*	$2f + 1$
Omission	f^*	∞	$2f + 1$	$2f + 1$	$2f + 1$
Byzantine	$3f + 1$	∞	$3f + 1$	$3f + 1$	$3f + 1$
Auth. Byzantine	f^*	∞	$3f + 1$	$2f + 1$	$3f + 1$

Table 3.2: The minimum number of nodes n, with at most f may have a faulty behavior, to solve the Byzantine generals problem under different time and execution assumptions. * the problem is considered as empty when $n \leq f + 2$.

3.3 Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) [24] is the first known implementation of a solution for the Byzantine generals problem. PBFT became the reference to achieve safety and liveness in partial synchrony by assuming that at most $\lfloor \frac{n-1}{3} \rfloor$ of nodes may be Byzantine. Note that, the partial synchrony corresponds to the first bullet of definitions defined in the previous section where the message delivery time is known and hold from some unknown point. Safety means that the nodes act as a centralized implementation that executes users operations atomically one at a time and liveness means that users eventually receive replies to their requests.

PBFT is divided in *view* and at each view a node plays the role of the leader. The view changes when it appears that the leader has failed. PBFT works roughly as follows. A user sends a request to invoke a service operation to the leader node. Then, the leader broadcasts the request to other nodes. Next, the leader and nodes execute the request and run a three-phase protocol to provide a reply to the user. At each phase, a node needs to reach a threshold of votes to execute the next phase. Finally, the user waits for $f + 1$ replies from different nodes with the same result after executing the operation to consider his request as confirmed.

3.4 Summary

Reaching a common agreement in distributed manner is not a new problem and comes from three closely related distributed agreement problems. The Byzantine generals problem has been considered as the pioneer problem since a solution to this problem solve the two other problems. In the Byzantine generals problem, a group of nodes tries to reach an agreement on a single value by assuming that some of them may be faulty, *a.k.a* Byzantine. A protocol that solves this problem has to satisfy the agreement, validity and termination properties.

Several theoretical results have been proved for the Byzantine generals problems by considering different timing assumptions. The first results led to the statement that *in a synchronous system without signed messages, it is possible to solve the Byzantine generals problem with* $n \geq 3f + 1$ *nodes with at most* f *nodes may be Byzantine.* The main impossibility result is the FLP impossibility that states that *it is impossible to have a solution to the consensus problem in an asynchronous system in which only one node may crash.*

Therefore, the timing assumption in the next works has been relaxed to obtain the assumption of partial synchrony that may have several meanings. The PBFT protocol became the reference to construct consensus protocols in a fixed, small and identified group of nodes. Indeed, PBFT achieves safety and liveness in partial synchrony by assuming that at most $\lfloor \frac{n-1}{3} \rfloor$ of nodes may be Byzantine.

Chapter 4

Consensus protocols for blockchain technologies

Contents

In this chapter, we give some examples of consensus protocols for blockchain and the first security analysis and formalization of these protocols.

4.1 Example of consensus protocols

This section presents relevant examples of consensus protocols to understand the main challenges to construct these protocols for blockchain technologies.

4.1.1 Proof-of-work (PoW) and new challenges

The concept of *Proof-of-Work*, abbreviated PoW, has been formalized by Jakobsson and Juels in 1999 [65]. The idea was as follows: *a prover demonstrates to a verifier that he has performed a certain amount of computational work in a specified interval of time.* This formalization was intended to outline PoW as a mechanism for several security goals, *e.g.* managing resource access, protection against spamming or other denial-of-service (DoS) attacks.

PoW in consensus protocol. In 2008, Nakamoto uses the PoW mechanism in the consensus protocol for the blockchain of Bitcoin [85]. Nakamoto uses a similar idea as the one formalized by Jakobsson and Juels [65]: the PoW mechanism is used in the leader election process where a node has to perform a difficult intensive computational work to be a candidate for the election.

The Nakamoto PoW protocol works as follows. A node who wants to be a candidate for the leader election has to perform a *mining*. That is, the node called *miner* forms a candidate block with valid transactions and a randomly selected value, *a.k.a nonce*, while solving a *hash puzzle*. The hash puzzle of Nakamoto consists to find a candidate block whose hash value is lower than a target value called *mining difficulty*. This latter is adjusted every 2016 blocks, *i.e.* every 14 days approximately, in order to maintain a block generation time close to 10 minutes. The leader who wins the right to write his candidate block into the blockchain is the first candidate who proposes a block with valid transactions and a hash value solving the hash puzzle. The block is considered as immutable when a majority of nodes votes for this block*. It is assumed that a majority of nodes votes for a block at height k when the block at height $k + 6$ is added into the blockchain. Finally, the leader receives his reward composed of the block reward and the transaction fees included in his block.

The PoW consensus protocol of Nakamoto can be seen as an election in which the node's probability of being elected as a leader is proportional to the computing/mining power owned by the node. The mining power generally refers to the number of computations to find a valid nonce in a given time. We say also that nodes are rewarded proportionally to their mining power.

New Bitcoin's blockchain requirements. Nakamoto's PoW protocol aims to achieve the new requirements of *scalability* and *incentivation* related to the Bitcoin blockchain's public aspect where anyone at any time can participate in the protocol.

The scalability enables to handle a variable number of nodes. Indeed, the open and dynamic participation of public blockchain does not allow to know the exact number of nodes who currently participate in the protocol. This requirement leads to achieve the same throughput when the number of nodes changes and in particular, when this number increases. The throughput is the time to execute a transaction, *i.e.* the time between the transaction is written into the ledger until it is considered as immutable to be executed.

The incentivization aims to motivate nodes to participate in the consensus protocol, *i.e.* verifying the transactions, adding new blocks of transactions and executing these valid transactions. For example, in the Bitcoin protocol, leaders are rewarded with some Bitcoin cryptocurrencies.

*More precisely, nodes with a majority of computing power vote for this block.

First Nakamoto PoW issues. However, the Nakamoto PoW protocol has a number of limitations such that requiring intensive energy consumption. Indeed, the hash puzzle can be only solved by brute force, *i.e.* testing all possible values of nonce until finding a hash value lower than the mining difficulty. This issue has led nodes to gather in *pools* in order to share the workload and smooth their revenue. In some ways, there is also a *centralization* [†] to big pools, *i.e.* the ones that gather more computing power. Thus, they have more chance to find a solution than small pools but each node of a pool wins proportionally to his mining power. The protocol is also vulnerable to several other issues such as fork problems leading to two valid blocks that extend the same block and compete to be the main chain or selfish mining strategies where the attacker temporarily hides one or several blocks in order to increase incomes by revealing her blocks at a suitable time. These issues and some others are detailed in Section 4.2.

4.1.2 Proof-of-Stake (PoS) and other mechanisms

Several alternatives have been mainly proposed to achieve the following goals: (i) preventing issues of the Nakamoto PoW protocol, (ii) satisfying safety and liveness properties and (iii) taking into account scalable and incentive needs.

First solutions have been based on a *Proof-of-Stake* (PoS) mechanism introduced as an energy-saving alternative to the Nakamoto PoW protocol. In PoS-based consensus protocols, the probability of being leader is proportional to the money invested and locked for the protocol, known as *stake*.

Peercoin [70] has been the first one that uses PoS mechanism in an hybrid PoW/PoS-based consensus protocol. The idea is that the mining difficulty of a node is proportional to his stake.

Ouroboros [68] proposed by Kiayias *et al.* is one of the first blockchain based on a *pure* PoS mechanism, *i.e.* without using PoW mechanism. At each round of Ouroboros, one election occurs and randomly selects a leader for each slot of the next round. The random seed for the election is computed via a multiparty computation protocol using publicly verifiable secret sharing [95].

Algorand [28, 53, 29] is also one of the first pure PoS-based protocols. At each round of Algorand, one or several potential leaders are randomly and privately selected. Indeed, their identity remains hidden until they decide to reveal themselves. Then, a rule selects one of them as leader. If there is not potential leader revealed, then the round is empty and a default block is added into the ledger. Algorand seems a promising approach to construct consensus protocols for blockchain and we study it in more detail in Chapter 6.

[†]Centralization issue may also be due to the industrialization of mining process with better-performing machines while seeking low-cost electricity.

PoS issues. Combining PoS and PoW mechanisms as in Peercoin, enables to partially solve the energy waste of PoW. Indeed, the hash puzzle still needs a brute force to find a solution and mostly computations performed by nodes are always wasted. PoS-based protocols may also lead to attacks such as *nothing-at-stake* due to the small computational effort for generating a block. For example, an attacker can invest currencies in several forks to increase her chance of being in the main chain and recover her currencies invested in the discarded chains. Another issue of PoS-based protocols is the centralization to the richer nodes since more money a node has invested in the protocol, more chance he has to be elected as leader.

Alternative solutions have been proposed to avoid these issues such as the Delegated Proof-of-Stake (DPoS) of Bitshare [14]. In a DPoS-based protocol, a set of validators is selected proportionally to the stake delegated by other nodes and takes turn in creating new blocks. If a validator is suspected of bad behavior, the delegator is punished.

Other mechanisms. Other solutions have been also proposed such as *Proof-of-Elapsed-Time* (PoET) [33, 19], *Proof-of-Authority* (PoA) [89] or *Single Secret Leader Election* (SSLE) [15]. In a PoET-based protocol, the leader is the node who gets the lowest *wait time* value randomly generated by a Trusted Execution Environment (TEE) that is a protected address space. In a PoA-based protocol, only a fixed and known set of nodes, called authorities, take turns in a predefined order in generating of blocks. In SSLE schemes, exactly one leader is randomly selected such that his identity is hidden until he decides to reveal it via an eligibility proof. The SSLE paper [15] presents a formal model with security properties that seems a relevant security analysis for leader election protocol and we study it in more details in Chapter 5.

Instead of replacing the PoW mechanism, some protocols aim to make the computations useful [66] by substituting the hash puzzle by another problem. For example, in Primecoin [69], the leader is the first participant who find a chain of prime numbers. Other mechanisms [8, 77] have been proposed to perform honest machine learning training work in order to have the chance to be the leader. The Folding@home [12] and SETI@home [3] projects aim to find solutions to real world problems such that researching various diseases and extra-terrestrial life respectively. Zhang *et al.* [103] proposed the *Resource-Efficient Mining* where one or several clients propose any useful works in form of tasks that nodes run in a TEE. Ball *et al.* proposed a *Proof-of-Useful-Work* (PoUW) [9] protocol to replace the Bitcoin PoW puzzle by the k-Orthogonal Vectors (k-OV) problem. Hybrid solutions [26, 58] have also been proposed to decrease the energy waste of mining process by combining the hash puzzle with useful works. We consider this topic as an interesting one and propose a new consensus protocol called Useful Work in Chapter 8.

4.2 Attacks and strategies

Several attacks already known in other systems have to be taken into account in the construction of consensus protocols for blockchain. For example, the denial of service (DoS) attack which makes the service of a system useless. Some security analysis have been done on blockchain and provide other new attacks, rational strategies and other problems. Table 4.1 summarizes some of these issues.

Name	*Overview*
Fork problem	Two valid branches extends the main chain.
Double spending attack	An attacker attempts to reuse the resources of a transaction for another purpose.
DoS or DDoS attack	An attacker floods other nodes to make it unavailable. In DDoS, the attack comes from multiple machines.
Sybil attack [39]	An attacker creates several malicious nodes in order to increase her influence on the protocol.
Majority attack	An attacker owns more than half of resources to participate in the protocol.
Grinding attack	An attacker cheats the leader election to increase her chance of being leader.
Long range attack [22, 100]	An attacker creates a longer valid chain starting from early block to alter the history.
Nothing-at-Stake attack	Developed against PoS-based protocols where an attacker may invest currencies in the system, runs an attack that is successful or not and recovers her currency.
Bribery attack [17, 100]	An attacker pays a node to work or discard specific block or chain that is beneficial for him
Selfish mining strategies [44]	An attacker hides temporarily one or several blocks and reveals them at the suitable time in order to increase her revenue.
Eclipse attack [60]	An attacker isolates a victim node from the rest of nodes to exploit him for her own interest.
Cloning attack [41]	An attacker partitions the network long enough to double spend.

Table 4.1: Attacks on consensus protocol and blockchain.

Fork problem

A fork in a blockchain occurs when two blocks extend the same block and compete to be in the main chain. This issue may be usual as in Nakamoto PoW protocol, where nodes can propose blocks at roughly the same time. Fork may also be caused by a malicious behavior. For example, an attacker can partition the network and create two or several groups of nodes that cannot communicate between them. Thus, each group may have a different view of the blockchain and continue to extend their blockchain view creating several different branches. An example of fork is done is Figure 4.1 where the two branches (B_3) and (B'_3, B'_4, B'_5) extend the block B_2.

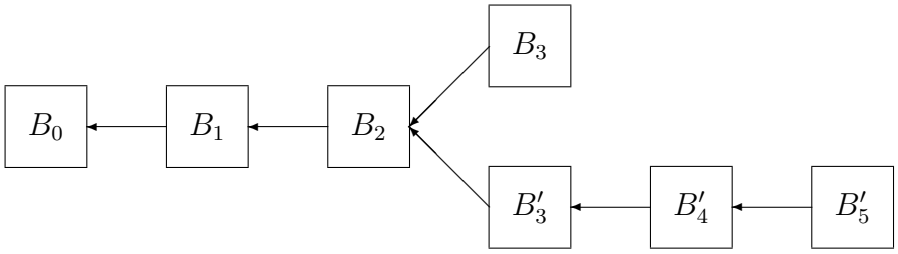


Figure 4.1: Fork.

The longest chain rule is generally used to solve the fork issue and aims to converge nodes in the same chain. This rule may state that the nodes have to choose the chain with the highest number of blocks or the chain with the more resources (computing power, money, etc.) invested in it. For example in Figure 4.1, the chain $(B_0, B_1, B_2, B'_3, B'_4, B'_5)$ may be chosen as the main chain. Other solutions may set some weight on blocks and the heaviest chain is chosen as the main chain.

When a fork is resolved, the transactions included in discarded blocks may return in the pool of pending transactions to be included in next blocks, as the transactions in block B_3 in Figure 4.1 for example.

Forks may delay the blockchain consistency and impact its performances such as delaying the validation of blocks and slowing down the throughput. For example, in the Nakamoto PoW protocol, a block may be considered as immutable after added 6 blocks (about 1 hour) to ensure that transactions included in this block can be safely executed. Moreover, a fork may facilitate an attacker to run another attack such as a double spending or selfish mining strategies as detailed in next sections.

Double spending attack

In digital currency systems, the double spending is one of the most known problems where an attacker attempts to reuse the currencies for another purpose. In a blockchain context, the double spending attack is successful when an asset already transferred via an executed transaction is reused for another purpose. For example, the attacker issues multiple conflicting transactions whose at least two are executed. The attacker can also succeed by reverting a transaction already executed and send assets in another transaction. `Verification of transactions` phase as defined in Chapter 2 may be an important phase to prevent the double spending attack.

DoS/DDoS attack

In a *denial of service* (DoS) attack, an attacker makes the network or service unavailable, for example, by flooding it of a huge amount of messages. In the case of *distributed* denial of service (DDoS), the attacker run a DoS from different sources.

For example, Conti *et al.* [32] present a DDoS attack where the adversary floods a node of *undecidable* messages. An undecidable message is a message that cannot be fully validated and has to be stored since it cannot be discarded. Thus, the victim node may be unavailable or isolated from other nodes since his memory and bandwidth is full.

This attack may impact the usefulness of a blockchain if leaders become unavailable to provide new blocks. For example, a cryptocurrency based on a useless blockchain where the participants can neither add nor execute new transactions may devalue this cryptocurrency. An attacker may purchase at low price the cryptocurrency in question. Then, she stops the DoS attack in order to the cryptocurrency value increases. Thus, the attacker may easily win money without investing more currencies in blockchain.

Sybil attack

In a *Sybil attack* [39], an attacker creates several malicious nodes under different identities to participate in the protocol. The goal can be to increase her influence in the protocol or to launch another attack such as a DDoS.

In a blockchain context, an attacker can create several nodes to verify, validate or drop transactions to benefit her transactions. The attacker may also create nodes to participate in the consensus protocol to approve blocks that are beneficial for her own interests, *e.g.* with a double spending attack.

Sybil nodes are generally mitigated with PoW or PoS mechanisms in a public blockchain. Indeed, splitting the resources (computing power or currencies) over several nodes may not impact the consensus protocol and the attacker has to invest

more resources to have more influence. In a private blockchain, Sybil nodes may be mitigated with the authentication of nodes.

Majority attack

Another well-known issue is the *majority attack* where an attacker owns $50\% + \alpha$, with $\alpha > 0$, of resources to participate in the protocol. Indeed, a such attacker in a blockchain as the one used in Bitcoin may perform all actions without notice from other nodes. For example, validating transactions and blocks that are beneficial for her or discarding data to make the blockchain useless.

Until July 2021 when Chinese government banned the mining farms in their country, Chinese pools had control more than 60% of mining power of Bitcoin system as outlined in graphs [20].

Grinding attack

In a *grinding attack*, the attacker attempts to influence the leader election process to increase her chance of being elected as leader. For example, an attacker may tamper with a random value used in a leader election process. In the blockchain context, this may centralize the protocol to the attacker that may be often elected as leader. This attack may be mitigated by guaranteeing a leader election process using a random source difficult to bias.

Long range attack

The purpose of the *long-range attack* [22] is to alter the history by creating a longer valid chain starting from an early block, even from the genesis block.

For example, the paper [100] presents a strategy enabling an attacker who corrupts a specific set of nodes to create a longer branch and fork the blockchain of Algorand. The attack is described in term of number of nodes that can be easily adapted in term of money. The strategy is the following. Let PK_k be the set of nodes that participate in the Algorand consensus protocol at the round k and $|PK_k|$ the number of nodes in the set PK_k. We denote B_k the block added during the round k. At the round k, we have the chain $(B_0, B_1, \ldots, B_{k_1}, \ldots, B_k)$. Assume that at a round $k_1 < k$, the set of nodes is $|PK_{k_1}| < \frac{1}{3}|PK_k|$. The attacker $\mathcal{A}$ chooses to corrupt the set of nodes PK_{k_1}. Thus, at round k the adversary $\mathcal{A}$ corrupts a total number $|PK_{k_1}|$ of nodes that is less than $\frac{1}{3}|PK_k|$. The adversary $\mathcal{A}$ can create a fork from the round k_1 until the round $k+1$. To this end, $\mathcal{A}$ generates the blocks $B'_{k_1}, B'_{k_1+1}, \ldots, B'_k$ with some transactions among the nodes in PK_{k_1}. For the block B'_{k+1}, the adversary $\mathcal{A}$ generates transactions with tiny amounts of money from the nodes in PK_{k_1} to the nodes in $PK_k \backslash PK_{k_1}$. In other words, the

nodes in $PK_k \backslash PK_{k_1}$ are included in the protocol from the block B'_{k+1} and so from the round $k+1$ instead of the round k with B_k. The blocks $B'_{k_1}, B'_{k_1+1}, \ldots, B'_{k+1}$ are easily confirmed since $\mathcal{A}$ controls all nodes in PK_{k_1} and the number $2n_v/3$ of votes is fixed. Thus, the chain $(B_0, \ldots, B_{k_1-1}, B'_{k_1}, B'_{k_1+1}, \ldots, B'_k, B'_{k+1})$ is longer than the chain $(B_0, B_1, \ldots, B_{k_1}, \ldots, B_k)$. Finally, the chain of the attacker is chosen as the main chain and the attack is successful.

A countermeasure would be adding checkpoints. They may guarantee that the blocks before a checkpoint can be considered as immutable. Thus, new blocks that modify the blocks before this checkpoint may be discarded.

Nothing-at-stake attack

The *nothing-at-stake* attack has originally targeted PoS-based consensus protocols since the generation of blocks necessitates little computational efforts. Indeed, an attacker may invest currencies in multiple chains of a fork by generating a block for each branch. Thus, it may guarantee that one of them is chosen to be in the main chain and the attacker can recover currencies invested in the discarded blocks. This attack could be possible in PoW-based protocols where a node can work on several forks. However, because of the need of significant mining power to generate a block, this attack could be not advantageous.

Bribery attack

In a *bribery* attack [17], an attacker pays a node to work on or discard specific blocks or chain. In consensus protocols, if the attacker pays a node more than his expected reward, then the node is encouraged to accept and work for the attacker, *i.e.* verifying, validating or discarding transactions or blocks as wanted by the attacker.

For example, in [100] the attacker asks to send the leader/verifier eligibility of Algorand participants before publicly revealing it or their keys to validate blocks since there is no incentive to keep this information secret. Thus, the attacker can target leaders and verifiers to corrupt in order that they produce data in her favor, such as including advantageous transactions for the attacker in new blocks or vote for a block to create a fork.

Selfish mining strategies

Selfish mining strategies [44] are ones of the first security analysis after the release of Bitcoin paper. These strategies originally target the Nakamoto PoW consensus protocol. In selfish mining strategies, a set of *selfish nodes* seen as an attacker hides temporarily one or several blocks to reveal them at the suitable time to increase

her revenue. Indeed, she takes advantage of the usual forking of the Nakamoto PoW protocol to mine a private chain that only selfish nodes work on. Then, this private chain is revealed judiciously to drop blocks honestly generated from the main chain. Selfish mining strategies are successful if the publication of the private chain constrains honest nodes to abandon the public branch, and thus to waste their computational power. In this way, the attacker can increase her ratio of blocks in the blockchain and also her reward compared to the reward she would obtain by following the honest protocol. Several strategies are proposed in [44]. A simplified mining algorithm is presented in Algorithm 1.

Let α the mining power of the selfish nodes set and γ the ratio of other nodes who receive firstly the selfish block and so mine on it (as in the strategy 2.(b) of Algorithm 1). The set of selfish nodes can obtain a revenue larger than their mining power when α satisfies the following range [44]:

$$\frac{1-\gamma}{3-2\gamma} < \alpha < \frac{1}{2}$$

The selfish mining is successful when the selfish nodes have at least 1/3 of the total mining power and so they can obtain more revenue than the expected ratio proportional to their mining power. Another relevant result is outlined in the Figure 4.2 [44]. If the selfish nodes may control a certain proportion of blocks propagation, then the selfish nodes can succeed the selfish mining with $\alpha \leq \frac{1}{3}$. For example, if the selfish nodes have $\alpha = \frac{1}{4}$, then they need that $\gamma = \frac{1}{2}$ of honest nodes mine for them, *i.e.* at least a majority of honest nodes receives firstly the selfish block. Figure 4.3 [44] plots the revenue of selfish nodes with different values of α

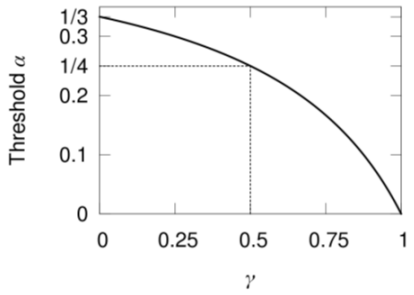


Figure 4.2: The ratio of honest nodes according to the mining power held by the selfish nodes to succeed the Selfish Mining attack in Bitcoin.

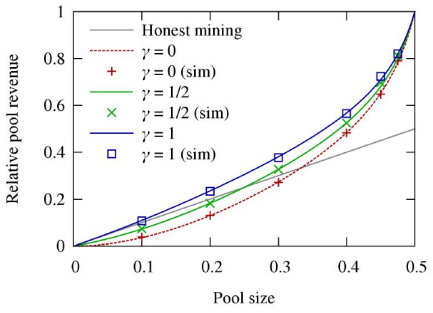


Figure 4.3: The revenue of the selfish nodes with different γ compared to the honest nodes revenue in Bitcoin.

Algorithm 1 Selfish mining strategy

The set of honest node HN and the set of selfish nodes SN start the mining on the same starting block $B_{l,k-1}$ generated by a node $\mathcal{P}_l$ at round $k-1$.
We denote by H a node in HN and by S a node in SN.
We denote by $B_{I,k}$ the block at round k generated by a node $I \in \{H, S\}$.

1. If H finds the next block, denoted by $B_{H,k}$, before SN

   $B_{H,k}$ is directly published since H honestly follows the protocol. SN starts again at the beginning of the selfish mining algorithm with $B_{H,k}$ as the starting block.
   Selfish mining is **not successful**.

2. Else, if S finds the next block, denoted by $B_{S,k}$, before HN

   S keeps this block private and continues to mine on $B_{S,k}$. Two cases are possible:

   (a) If S finds the next block $B_{S,k+1}$:

   SN gets ahead of HN and publishes one block at each time H finds a new block.

   Selfish mining is **successful** because SN continues to mine on her private chain whereas HN has to start again their mining at each block published by a selfish node. Thus SN earns all the revenues of her valid blocks that she publishes.

   (b) Else, H finds also the next block $B_{H,k}$:

   S publishes instantly her block $B_{S,k}$.

   A part of HN receives firstly the block $B_{H,k}$ and the other part of HN receives firstly the block $B_{S,k}$. Three cases are possible:

   i. S finds the block $B_{S,k+1}$ and publishes it:

   SN earns the reward of blocks $B_{S,k}$ and $B_{S,k+1}$.
   Selfish mining is **successful**.

   ii. H finds the block $B_{H,k+1}$ mined on the block $B_{S,k}$:

   HN earns the reward of block $B_{H,k+1}$ and SN earns the reward of $B_{S,k}$.
   Selfish mining is **successful**.

   iii. H finds the block $B_{H,k+1}$ mined on their block $B_{H,k}$:

   HN earns the rewards of their blocks $B_{H,k}$ and $B_{H,k+1}$.
   Selfish mining is **not successful**.

and γ. In particular, with $(\alpha, \gamma) \geq (\frac{1}{3}, 0)$ and $(\alpha, \gamma) \geq (\frac{1}{4}, \frac{1}{2})$, the revenue obtained with the selfish mining is higher than the expected revenue earned by following honestly the protocol.

A possible countermeasure to limit selfish mining strategies would be to change the rule of mining on the first block received and the longest chain choice in the case of a fork. For example, as proposed in [44], nodes can collect chains of same length and choose uniformly at random one to mine the next block.

Eclipse attack

Eclipse attack [17] enables to isolate a victim node from the rest of the network to exploit him for her own interest, such as filtering the node's view of the blockchain, forcing to waste computational power or using the node's computing power. The main idea is to control a sufficient number of IP addresses to monopolize incoming and outgoing connections of the victim. With a certain number of addresses, an eclipse attack can be successful with a probability of 85% [17]. Several countermeasures are proposed in [17] such as a new mechanism to fill the addresses tables to prevent attacker addresses to be added into tables.

An eclipse attack can be useful for selfish mining strategies for example. Indeed, selfish nodes can increase the ratio of honest nodes who mine for selfish nodes, denoted by γ in selfish mining strategies. Thus, blocks found by the eclipsed nodes may be not forwarded to honest nodes and only blocks generated by selfish nodes may be provided to eclipsed nodes. Note that, a node (specifically for Bitcoin and public blockchain) may have several public addresses and an attacker has to determine which ones own to the victim node. An eclipse attack is successful if each victim node's public address is eclipsed from others.

Cloning attack

Cloning attack [41] aims to perform a double spending attack in a Proof-of-Authority (PoA) consensus protocol. In a PoA-based protocol, a set of authorities takes turn in predefined order to generate new blocks. In a cloning attack, an attacker duplicates her instance in a clone which uses her pair of public and private keys to generate and vote for blocks. By partitioning participants into two groups that cannot communicate between each other, an attacker and her clone can participate in the chain of both groups. The attacker and her clone provide conflicting transactions to their respective group. The goal is to 1) confirm the two transactions by maintaining enough time the partitioning and 2) create a longer chain in one of the groups in order to discard the chain of the other group. Thus, when the partitioning is stopped, the conflicting transactions are confirmed but only one chain is selected as the main chain due to the longest chain rule.

The conditions proposed in [41] to prevent the cloning attack in a PoA-based protocol is $\frac{n+f}{2} < |V| < n - f$ where V the set of authorities to confirm a block and f is the number of malicious ones.

4.3 First formalization

In the original paper of Bitcoin [85], Nakamoto proposes a solution to solve both the consensus problem and the double spending issue assuming open participation without a trusted third party. However, the paper [85] does not provide formal proof of this claim. Table 4.2 lists some of the first formalizations of the Nakamoto PoW protocol.

Date	*Title*	*Overview*
2014	Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin [81]	A first formal definition of Bitcoin in a synchronous model
2014	The Bitcoin Backbone Protocol: Analysis and Application [47] [49, 48]	First formalization of fundamental properties of Bitcoin protocol in a synchronous model with an adver--sary with less than half of total power.
2016	Analysis of the Blockchain Protocol in Asynchronous Networks [86]	Extend [49] in partially synchrony.
2016	The Bitcoin Backbone Protocol with Chains of Variable Difficulty [50]	Extend [49] in dynamic and partially synchronous setting.
2017	A formal model of Bitcoin transactions [5]	Prove the properties of no double spending and non-increasing value.

Table 4.2: First formalization of the Nakamoto PoW protocol.

In [47], Garay *et al.* formalize the core behind the Nakamoto PoW protocol, namely *Bitcoin backbone*. It is defined by these three following properties:

- **common-prefix** with parameter $k \in \mathbb{N}$: it holds that the probability for any two honest nodes maintain the same prefix of chain by removing k blocks from the end of their local chain increases exponentially in k.
- **chain-quality** with parameters $l \in \mathbb{N}$ and $\nu \in \mathbb{R}$: it holds that in any l

consecutive blocks of a chain of any honest node, the ratio of honest blocks is at least ν.

- **chain-growth** with parameters $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$: after s consecutive rounds, it holds that any honest node adopts a chain that is at least $\tau \cdot s$ blocks longer than his local chain.

To the best of our knowledge, their paper [47] and its updated versions [49, 48] can be considered as the first ones that present the best formalization with a security proof in the domain of consensus protocols for blockchain. Indeed, all steps to provide a security proof are respected: formalization of consensus protocols, formalization of the adversary power, formalization of security properties and finally a proof of these properties. Thereafter, works providing security proofs of consensus protocols are mainly based on the Bitcoin backbone model.

4.4 Summary

The well-known PoW consensus protocol proposed in the Bitcoin blockchain by Nakamoto elects in distributed manner a leader proportionally to his computing power to generate the next block to be added into the ledger. The Nakamoto PoW consensus protocol aims to solve the consensus problem while satisfying the scalability and incentivation requirements of the public blockchain.

However, several issues have not been taken into account such as the fork problem, the centralization in big pools, the selfish mining strategies and many others as seen in this chapter. Several alternatives have been proposed to prevent Nakamoto PoW issues based on mechanisms such as PoS, PoET, PoA and many others [101]. New security analysis have outlined other issues such as the centralization in richer nodes, the nothing-at-stake issue, the cloning attack, etc. Some formalization have also been published and the paper on the Bitcoin backbone became the influential work to analyze consensus protocols for blockchain.

With all these works, we observe that the main process in a consensus protocol is the leader election one that elects randomly and in distributed manner a leader to generate the next block to be added into the ledger. Thus, we focus our work on this process in the three following chapters.

Chapter 5

Revisiting security analysis of Single Secret Leader Election

Contents

5.1 SSLE overview

Recently, a formal model of *Single Secret Leader Election (SSLE)* scheme was introduced in [15]. In a SSLE protocol, a group of n participants aim to randomly choose exactly one leader such that the identity of the leader will be only known by the chosen leader. Later, the elected leader can reveal his identity while proving that he indeed won the election.

Three constructions are provided in [15]. A first construction is built upon indistinguishability obfuscation [10, 52] and a second construction is built upon threshold FHE [16]. These constructions aim to show the feasibility of constructing a SSLE protocol and do not target real-life applications. Then, the third construction called *shuffling-based SSLE* is built upon the hardness of the Decisional Diffie-Hellman (DDH). This shuffling-based SSLE scheme that we study in this chapter has been proposed as a more suitable construction for practical applications.

A SSLE scheme must meet three properties: (i) *uniqueness*: exactly one leader is chosen in each election; (ii) *fairness*: each entity has a probability $\frac{1}{n}$ of becoming the leader; and (iii) *unpredictability*: an adversary who does not control the leader cannot learn which entity has been elected before the leader decides to reveal himself.

5.2 Security model

In a SSLE [15] protocol, a group of at most N participants $\mathcal{P}_1, \ldots, \mathcal{P}_N$ repeatedly elects exactly one leader $\mathcal{P}_i$. At every election, each participant learns whether he is the leader and if yes, he can provide a proof that she was elected.

5.2.1 SSLE model

Let λ be the security parameter. We assume that a fresh public randomness Q is generated for each election. We adapt the original definition of a SSLE scheme provided in [15]. Indeed, we simplify the model to fit better with the construction based on the hardness of Decisional Diffie-Hellman that we focus on in this chapter. A SSLE protocol is defined as a tuple of Probabilistic Polynomial Time (PPT) algorithms $\texttt{SSLE} = (\texttt{SSLE.Setup}, \texttt{SSLE.KeyGen}, \texttt{SSLE.Register}, \texttt{SSLE.RegisterVerify}, \texttt{SSLE.Elect}, \texttt{SSLE.Verify})$.

The algorithm `SSLE.Setup` is a process intended to be run a single time before initiating a series of elections:

- $\texttt{SSLE.Setup}(1^\lambda, N) \rightarrow \texttt{pp}, \texttt{st}_0$. On input 1^λ and the maximum number of participants N, it generates the public parameters `pp` and the initial state $\texttt{st}_0$.

The algorithm `SSLE.KeyGen` is run by each participant before executing the registration process:

- `SSLE.KeyGen` $(1^\lambda$,`pp`$)\rightarrow sk, pk$. On input 1^λ and the public parameters `pp`, it generates a secret key sk and a public key pk.

The process `SSLE.Register` is run by each participant when he wants to start taking part in elections:

- `SSLE.Register`$(i$,`pp`,`st`,$sk_i, pk_i) \rightarrow p_i$,`st'`. Each participant registers to take part in elections using a unique public identity $i \in \{1, \ldots, N\}$, the public parameter `pp`, the current state `st`, the secret key sk_i and the public key pk_i. Registration outputs an index of registration p_i for a value v_i* that is computed using the secret value sk_i and it modifies the state `st` to `st'` by adding the pair (i, pk_i) to the state `st` and by shuffling the list of values v_j's such that v_i is ranked p_i.

The participant registers once and stays registered unless he decides to leave or he has been elected. Indeed, an elected leader has to re-register after having been elected.
Every time a new participant has registered, each previously registered participant has to run the process `SSLE.RegisterVerify` in order to verify that the registration was carried out correctly:

- `SSLE.RegisterVerify`$(i$,`pp`,`st`,$sk_i, p_i) \rightarrow 0/1$. Verification of registration is done using participant's secret key sk_i, the index of registration p_i, the public parameters `pp` and the current state `st`. It outputs 1 if the new participant has registered correctly from the perspective of participant $\mathcal{P}_i$, meaning that $\mathcal{P}_i$ is able to retrieve his randomized value v_i in his related bucket, and 0 otherwise.

The process `SSLE.Elect` is run by each participant when he wants to check whether he is elected as the leader for the current election:

- `SSLE.Elect`$(i$,`pp`,`st`,$Q, sk_i, p_i) \rightarrow (0, \perp)/(1, \pi)$. Leader election begins by taking as inputs the public parameters `pp`, current state `st`, and a random beacon $Q \in \mathcal{R}$, and it outputs whether participant $\mathcal{P}_i$ has been chosen as the leader. In case the participant $\mathcal{P}_i$ has been chosen as the leader, it also outputs a proof of leadership computed using sk_i.

The process `SSLE.Verify` is used to check whether a participant who claims to be the leader, when it is time for the leader to reveal himself, is indeed elected leader.

*We add this value in the model to be more accurate than [15]

- `SSLE.Verify(`i`, pp, st, `Q`, `π`)`$\rightarrow 1/0$. Given an index i, the state `st`, the election randomness Q, a proof π claiming that a particular participant $\mathcal{P}_i$ was elected leader, the verification algorithm accepts or rejects the proof that the participant $\mathcal{P}_i$ has been elected leader.

5.2.2 SSLE security properties

Before describing the security properties of a SSLE scheme, we provide a generic experiment that applies for the three defined security properties.

Generic experiment

It is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Setup Phase:** The adversary $\mathcal{A}$ selects the maximum number of participants N. The challenger $\mathcal{C}$ runs `SSLE.Setup` $(1^\lambda, N)$ and it outputs `(pp,st`$_0$`)`. The adversary $\mathcal{A}$ selects the number of corrupted participants $f < N$ and a related subset M of indexes in $\{1, \ldots, N\}$ of size f. The challenger $\mathcal{C}$ executes `SSLE.KeyGen(`1^λ`,pp)` for the uncorrupted participants and gets corresponding values (sk_i, pk_i).

- **Election Phase:** The challenger $\mathcal{C}$ generates an election randomness Q_0. To register an uncorrupted participant, the adversary $\mathcal{A}$ sends the index i of the participant to the challenger $\mathcal{C}$. Then, the challenger $\mathcal{C}$ runs `SSLE.Register(`i`,pp,st,`sk_i, pk_i`)` and it gives back to the adversary $\mathcal{A}$ the values pk_i, p_i and `st'`. To register a corrupted participant, the adversary $\mathcal{A}$ sends the index i of the participant to the challenger $\mathcal{C}$ with an updated state `st'`. In either case, the challenger $\mathcal{C}$ runs `SSLE.RegisterVerify(`j`,pp,st',`sk_j, p_j) for any previously registered participant $\mathcal{P}_j$ which is an uncorrupted participant. If it ever happens that there is a call to `SSLE.RegisterVerify(`j`, pp,st',`sk_j,p_j) that returns 0, then the game immediately ends with output 0. Otherwise, the state is updated to `st'`. The number of registered participants is denoted by n with $n \leq N$.
  During this phase, several elections may occur. For each new election, the challenger first generates a new randomness Q. Then, the challenger $\mathcal{C}$ executes on behalf of each uncorrupted participant $\mathcal{P}_i$ the algorithm `SSLE.Elect(`j`,pp,st,`Q, sk_j, p_j`)` and $\mathcal{C}$ sends the output $(1, \pi_j)$ or 0 to $\mathcal{A}$.

After the election phase, a challenge phase is executed. This latter phase is specifically defined for each property.

Uniqueness

The *uniqueness* property requires that exactly one leader is chosen in each election. It means that exactly one participant in an election can prove that he is the elected leader. This property is related to the safety property and more specifically to the following requirement: only a single value is chosen. We recall the experiment of the uniqueness property which is the same as defined in [15].

It is assumed that an adversary $\mathcal{A}$ can corrupt as many participants as she wants denoted by f with $f \leq n$. Since a corrupted participant that is elected leader may choose not to announce that she is the leader, it is possible that zero leader is elected.

We denote the uniqueness experiment by $\texttt{UNIQUE}[\mathcal{A}, \lambda, N, f, n]$. The experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** For the election with randomness Q, the adversary $\mathcal{A}$ outputs values (b_i, π_i) for each corrupted participant. The experiment outputs 0 if for the election with randomness Q and state $\texttt{st}$, there is at most one participant $\mathcal{P}_i^*$ that outputs $b_{i^*} = 1$ and π_{i^*} such that $\texttt{Verify}(i^*, \texttt{pp}, \texttt{st}, Q, \pi_{i^*}) = 1$. Otherwise, the experiment outputs 1.

We say that a SSLE scheme $\mathcal{S}$ is *unique* if no PPT adversary $\mathcal{A}$ can win the uniqueness game except with negligible probability. That is, for all PPT $\mathcal{A}$, we have:

$$Pr\Big[\texttt{UNIQUE}[\mathcal{A}, \lambda, N, f, n] = 1\Big] \leq \texttt{negl}(\lambda)$$

Fairness

The property of fairness means that each participant has a probability $\frac{1}{n}$ of being elected as the leader. However, the formal definition provided in [15] does not fit exactly with this informal definition. Indeed, the authors consider that as long as the probability for the adversary to be the elected leader is equal to the fraction of adversary-controlled participants, the protocol is fair. We disagree with this global definition of probability of success since, in a fair leader election process, even in the set of corrupted participants, the probability that a specific corrupted participant is elected should be $\frac{1}{n}$. Indeed, in some leader election protocols like Algorand [29] or Fantomette [7], the selection of a specific participant for a given election has an impact on following elections. This may also happen in the SSLE assuming that the randomness Q is generated from the history of previous elections. We discuss this remark in more detail in Section 5.6. Consequently, we adapt the fairness game originally described in [15] in order to capture this broader notion of fairness.

It is assumed that an adversary can corrupt as many participants as she wants denoted by f with $f \leq n$. We denote the fairness experiment by $\texttt{FAIR}[\mathcal{A}, \lambda, N, f, n]$. The experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** At some point, all participants have registered. Then, the adversary $\mathcal{A}$ selects a participant i and one more election occurs. The experiment $\texttt{FAIR}[\mathcal{A}, \lambda, N, f, n]$ outputs 1 if $\texttt{Verify}(i,\texttt{pp},\texttt{st},Q,\pi_i)= 1$ in the challenge election. Otherwise, it outputs 0.

We say that a SSLE scheme $\mathcal{S}$ is fair if no PPT adversary $\mathcal{A}$ can win the fairness game with greater than negligible advantage. That is, for all PPT $\mathcal{A}$, we have:

$$\left| Pr\left[\texttt{FAIR}[\mathcal{A}, \lambda, N, f, n] = 1\right] - \frac{1}{n} \right| \leq \texttt{negl}(\lambda)$$

Unpredictability

The property of unpredictability means that an adversary $\mathcal{A}$ who does not control the leader cannot learn which participant has been elected. The unpredictability game has been originally defined to avoid the case where there is no uncorrupted participant in the bucket b^* from which the winner of the election is elected. In that case, $\mathcal{A}$ controls the winner of the election and then it is possible for $\mathcal{A}$ to predict the winner. The following case has also to be taken into account: when there is exactly one uncorrupted participant into the bucket b^*, then $\mathcal{A}$ can predict with probability 1 who is the winner of the election. We thus adapt the unpredictability game originally defined in [15] to prevent this specific case where $\mathcal{A}$ has the capability to learn which participant has been elected.

It is assumed that $\mathcal{A}$ can corrupt as many participants as she wants denoted by f with $f \leq n-2$. We denote the unpredictability experiment by $\texttt{UNPRED}[\mathcal{A}, \lambda, N, f, n]$. The experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** At some point, all participants have registered. Then, the adversary $\mathcal{A}$ indicates that she wishes to receive a challenge and one more election occurs. In this election, the challenger $\mathcal{C}$ does not send (b_j, π_j) for each uncorrupted participant to the adversary $\mathcal{A}$. Let $\mathcal{P}_i$ be the winner of this election. The game ends with the adversary $\mathcal{A}$ outputting an index i' in $\{0, \ldots, N\}$. If $\mathcal{P}_i$ is a corrupted participant or if $\mathcal{P}_i$ is the single uncorrupted participant in his bucket b^* then the output of $\texttt{UNPRED}[\mathcal{A}, \lambda, N, f, n]$ is set to 0. Otherwise $\texttt{UNPRED}[\mathcal{A}, \lambda, N, f, n]$ is set to 1 if and only if $i = i'$. By default, the output is 0.

We say that a SSLE scheme $\mathcal{S}$ is unpredictable if no PPT adversary $\mathcal{A}$ can win the unpredictability game with greater than negligible advantage when the winner of

the election is uncorrupted. That is, for all PPT $\mathcal{A}$, for any $f \leq n-2$ and $n \leq N$, we have:

$$Pr\Big[\texttt{UNPRED}[\mathcal{A}, \lambda, N, f, n] = 1 \quad | \quad i \in \{1, \ldots, N\} \setminus M\Big] \leq \frac{1}{n^*} + \texttt{negl}(\lambda)$$

where n^* is the number of uncorrupted participants in the bucket from which the elected participant is selected.

Liveness

The liveness property is one basic security property of consensus protocols [73, 74] and also leader election. The liveness property as defined in [7] requires that even if a fraction of participants is controlled by an adversary $\mathcal{A}$, it is still possible to elect a leader. This informal definition means that at each election, at least one participant provides a valid leader eligibility proof that is accepted by the other participants. Thus, the liveness property guarantees that there is at least one leader at each election. For instance, in a context of blockchain, this enables also new blocks of data generated by leaders to be continually added into the ledger. Therefore, we formally describe the liveness property which is not defined in the original security model of a SSLE scheme [15].

The liveness experiment is denoted by $\texttt{LIVE}[\mathcal{A}, \lambda, N, n, f]$ and is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge phase.** For the election with randomness Q, the adversary $\mathcal{A}$ outputs values (b_i, π_i) for each corrupted participant. The experiment outputs 0 if for the election with randomness Q and state $\texttt{st}$, there is at least one participant $\mathcal{P}_i^*$ that outputs $b_{i^*} = 1$ and π_{i^*} such that $\texttt{Verify}(i^*, \texttt{pp}, \texttt{st}, Q, \pi_{i^*}) = 1$. Otherwise, the experiment outputs 1.

We say that a SSLE scheme is *live* if no PPT adversary $\mathcal{A}$ can win the liveness game except with negligible probability. That is, for all PPT $\mathcal{A}$ we have:

$$Pr[\texttt{LIVE}[\mathcal{A}, \lambda, n, c] = 1] \leq negl(\lambda)$$

5.3 Shuffling-based SSLE construction

Hereinafter, we recall the description of the shuffling-based SSLE construction based on DDH introduced in [15].

Let $\mathbb{G}$ be a group of prime order q where DDH is hard. Let H be a hash function.

- $\texttt{SSLE.Setup}(1^\lambda, N)$: select a public generator $g \in_R \mathbb{G}$ and set $\mathcal{L} = \{\}$. Output $\texttt{pp} = (g, N)$ and $\texttt{st}_0 = \mathcal{L}$.

- `SSLE.KeyGen(`1^λ`,pp)`: select a random value $k_i \in_R \{0,1\}^\lambda$. Compute $(dk_i, pk_i) = H(k_i)$. The public key is pk_i and the secret key is $sk_i = (k_i, dk_i)$.

- `SSLE.Register(`i`,pp,st,`sk_i, pk_i`)`. The current state `st` can be parsed as follows: `st`$= (\mathcal{L}, pk_1, \ldots, pk_{(i-1)})$.

  - Make a first update of the state `st` by adding pk_i to the list $pk_1, \ldots, pk_{(i-1)}$.
  - Then, select a random value $r_i \in_R \mathbb{Z}_q$. If there is any entry at index p_i in $\mathcal{L}$ whose value is $\perp$, then replace it by $(g^{r_i}, g^{r_i dk_i})$. Else, append the pair $(g^{r_i}, g^{r_i dk_i})$ to $\mathcal{L}$ and set $p_i = |\mathcal{L}|$.
  - Next, sample a random permutation Π on $\lceil\sqrt{N}\rceil$ elements and set $b = p_i \mod \sqrt{N}$.
  - Finally, update $\mathcal{L}$, and thus the current state `st`, such that each entry $\mathcal{L}_{j\cdot b} = (u_{j\cdot b}, v_{j\cdot b})$ is replaced by $\mathcal{L}_{j\cdot b} = (u^{r_j}_{\Pi(j)\cdot b}, v^{r_j}_{\Pi(j)\cdot b})$ for $r_j \in_R \mathbb{Z}_q$, where $\mathcal{L}_{j\cdot b}$ denotes the entries in $\mathcal{L}$ at index $j = b \mod \sqrt{N}$ for $1 \leq j \leq |\mathcal{L}|$.

  The output is $(p_i, \mathtt{st}')$ with p_i the updated index where the participant's new entry has been moved by Π.

- `SSLE.RegisterVerify(`i`,pp,st,`dk_i, p_i`)`. Parse the current state `st` $= (\mathcal{L}, pk_1, \ldots, pk_n)$ and recover the value b of the newly registered participant. Then, run the following checks:

  - If $p_i = b \mod \sqrt{N}$, then the newly registered participant is in the same bucket as $\mathcal{P}_i$ and he has to check that there is exactly one entry $\mathcal{L}_j = (u_j, v_j)$ in the bucket where j is a multiple of $p_i \mod \sqrt{N}$ such that $u_j^{dk_i} = v_j$ and update $p_i = j$ for that entry.
  - If $p_i \neq b \mod \sqrt{N}$, then the newly registered participant is not in the same bucket as $\mathcal{P}_i$ and he has to check that there is no entry $\mathcal{L}_j = (u_j, v_j)$ in the bucket where j is a multiple of $b \mod \sqrt{N}$ such that $u_j^{dk_i} = v_j$.
  - Check that there are no duplicates among $pk_1, \ldots, pk_n$.

  If the above checks pass, then the output is 1. Otherwise, the output is 0.

- `SSLE.Elect(`i`,pp,st,`Q, k_i, p_i`)`. Parse the state `st` $= (\mathcal{L}, pk_1, \ldots, pk_n)$. Let z be the number of times $\perp$ appears in $\mathcal{L}$ and let z' be the number of times $\perp$ appears before the i^{th} entry of $\mathcal{L}$. If $p_{i-z'} \neq Q \mod (N - z)$, output 0. Otherwise, remove entry $\mathcal{L}_{p_i}$ from $\mathcal{L}$ and pk_i from `st`, set $\pi = (i, p_i, k_i)$ and output 1.

- `SSLE.Verify(`i`,pp,st,`Q, π`)` Parse `st` $= (\mathcal{L}, pk_1, \ldots, pk_n)$, $\pi = (i, p_i, k_i)$ and $\mathcal{L}_i = (u, v)$. Compute $(dk'_i, pk'_i) = H(k_i)$. If $p_{i-z'} = Q \mod (N - z)$, $u^{dk'_i} = v$ and $pk'_i = pk_i$, output 1. Otherwise, output 0.

The next election is then run with the updated $\mathcal{L}$ where the entry $\mathcal{L}_{p_i}$ has been removed and entries of new registered participants have been added.

5.4 Security analysis

In this section, we analyze the shuffling-based SSLE scheme described in Section 5.3. For each security property, either we prove that the shuffling-based SSLE scheme satisfies the security property or we provide a strategy that breaks the security property.

5.4.1 Uniqueness: security proof

The uniqueness property described in Section 5.2 has not been modified from the original one definition of [15]. Thus, we do not recall the proof for the uniqueness property which is exactly the same as the one provided in [15] that proves the following theorem.

Theorem 1 *For any adversary $\mathcal{A}$, the shuffling-based SSLE construction provided in Section 5.3 is a unique SSLE scheme in the random oracle model under the assumption that $\mathbb{G}$ is a group in which the DDH problem is hard.*

5.4.2 Fairness: a strategy to introduce a bias

The fairness property of a SSLE protocol informally defined as, each participant having a probability $\frac{1}{n}$ of becoming the leader should enable, as an example, to organize bets on who will be elected. In this case, an adversary could attempt to introduce a bias into the election process in order to elect one specific corrupted participant. In this section we explain the strategy enabling to introduce such a bias in the leader election process. We explain how to detect it in Section 5.5.

Let $\mathcal{A}$ be a collaborative set of participants that want to make elected one specific corrupted participant $\mathcal{A}_h$ with $h \in \{1, \ldots, t\}$. For the sake of simplicity, let us assume that $\mathcal{A}_1, \ldots \mathcal{A}_t$ have been affected to the same bucket. During the registration process, every $\mathcal{A}_i$ may follow the regular procedure or modify the values of some or all collaborative participants already registered in the same bucket. Since the security of the scheme proposed in [15] relies on the check by every participant that he still finds his value in $\mathcal{L}$, then $\mathcal{A}_i$ can do the following without being detected at this stage by honest participants:

- Parse $\mathtt{st}= (\mathcal{L}, pk_1, \ldots, pk_{(i-1)})$.
- Select a first random value $k_i \in_R \{0,1\}^\lambda$ and compute $(dk_i, pk_i) = H(k_i)$. Set $sk_i = (k_i, dk_i)$ and make a first update of the state $\mathtt{st}$ by adding pk_i to the list $pk_1, \ldots, pk_{(i-1)}$. This step is executed in a regular way.
- Select a second random value $r_i \in_R \mathbb{Z}_q$. If there is any entry at index p_i in $\mathcal{L}$ whose value is $\perp$, then replace it by $(g^{r_i}, g^{r_i dk_1})$. Else, it appends the pair $(g^{r_i}, g^{r_i dk_1})$ to $\mathcal{L}$ and set $p_i = |\mathcal{L}|$. Note that, in this example, $\mathcal{A}_1$ has shared the knowledge of the secret value dk_1 with $\mathcal{A}_i$.
- Sample a random permutation Π on $\lceil \sqrt{N} \rceil$ elements and set $b = p_i \mod \sqrt{N}$.
- Finally, update $\mathcal{L}$, and thus the current state $\mathtt{st}$, such that each $\mathcal{L}_{j \cdot b} = (u_{j \cdot b}, v_{j \cdot b})$ is replaced by $\mathcal{L}_{j \cdot b} = (u_{\Pi(j) \cdot b}^{r_j}, v_{\Pi(j) \cdot b}^{r_j})$ for $r_j \in_R \mathbb{Z}_q$.

The output is $(sk_i, pk_i, p_i, \mathtt{st}')$ with p_i the updated index where the participant's new entry has been moved by Π.

When f collaborative participants are in the same bucket, they can all commit on the same value dk_1 for example by selecting different values r_i's, and on different values pk_i without being detected during the registration phase. Thus, in the bucket there are f values that commit on the same value dk_1. The size of the bucket is $\frac{|\mathcal{L}|}{\sqrt{N}}$. The probability that $\mathcal{A}_1$ is elected is $\frac{f}{|\mathcal{L}|}$ instead of a uniform probability $\frac{1}{|\mathcal{L}|}$.

5.4.3 Unpredictability: security proof

In this section, we provide the proof of the following theorem.

Theorem 2 *Under the assumption that $\mathbb{G}$ is a group in which DDH problem is hard, then for any adversary $\mathcal{A}$, the shuffling-based SSLE construction provided in Section 5.3 is a $\left(\frac{1}{n*} + \mathtt{negl}(\lambda)\right)$-unpredictable SSLE scheme in the random oracle model where n^* is the minimum number of uncorrupted participants in buckets containing uncorrupted participants.*

Proof. The proof for unpredictability is an adaptation of the proof provided in [15] by taking into account that there should be at least 2 honest participants in buckets containing honest participants. The unpredictable proof is done through a series of hybrid experiments.

- $\mathtt{H}_0[x]$: the real unpredictability game $\mathtt{UNPRED}[\mathcal{A}, \lambda, N, f, n]$ with the following additional abort condition. Le b^* be the bucket from which the winner of the challenge election is chosen. The game aborts if the x^{th} registration is not the last registration of an uncorrupted participant into b^* before the challenge

election. Note that, while there is at least two uncorrupted participants in b^* during the challenge election, such a registration will always exist. Otherwise, the winner will not be an uncorrupted participant or the winner of the challenge election is predictable with probability 1 if there is a single uncorrupted participant registered into b^*.

- $\mathtt{H}_1[x]$: the same as experiment $\mathtt{H}_0[x]$ except that instead of the winner being defined by the participant that can produce a proof of leadership that will be accepted by `SSLE.TVerify`, the winner is defined to be the participant $\mathcal{P}_i$ for which $u^{dk_i} = v$ where $(u, v) \in \mathcal{L}$ is the entry chosen by the election randomness Q.

Since the two definitions are equivalent, $\mathtt{H}_1[x]$ is indistinguishable from $\mathtt{H}_0[x]$.

- $\mathtt{H}_2[x]$: the same as experiment $\mathtt{H}_1[x]$ except that the experiment outputs 0 if the adversary ever queries the random oracle on the secret k_i of an uncorrupted participant who participates in the challenge election.

Any values of (dk_i, pk_i) belonging to an uncorrupted participant $\mathcal{P}_i$ appear inde--pendently random to the adversary until k_i is revealed to prove that $\mathcal{P}_i$ has been elected leader, unless the adversary queries H at k_i with probability at most $\frac{q}{2^\lambda}$ if it makes q queries. The probability that the adversary queries H at a point corresponding to any uncorrupted participant's secret is at most $\frac{Nq}{2^\lambda} \leq \mathtt{negl}(\lambda)$. Then, no PPT adversary could distinguish between $\mathtt{H}_1[x]$ and $\mathtt{H}_2[x]$.

- $\mathtt{H}_3[x]$: in this experiment, the challenger chooses the value of Q to be used in the election challenge during the setup phase instead of during the challenge phase so that the challenger knows at setup time which bucket the leader will be chosen from in the election challenge.

In $\mathtt{H}_3[x]$, the distribution of messages sent by the challenger is not modified and it is identical to $\mathtt{H}_2[x]$ in terms of adversary's view. Then, no PPT adversary could distinguish between $\mathtt{H}_2[x]$ and $\mathtt{H}_3[x]$.

- $\mathtt{H}_4[x]$: the same as in experiment $\mathtt{H}_3[x]$ except that the challenger is behaving differently in the x^{th} registration. During this registration, instead of replacing each $\mathcal{L}_{j\cdot b^*} = (u_{j\cdot b^*}, v_{j\cdot b^*}) = (u_{j\cdot b^*}, u_{j\cdot b^*}^{dk_i})$ by $\mathcal{L}_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{dk_i r_j})$ for $r_j \leftarrow \mathbb{Z}_q$ for entries corresponding to the secret dk_i of uncorrupted participant $\mathcal{P}_i$, it sets $\mathcal{L}_{j\cdot b^*} \leftarrow (u_{\Pi(j)\cdot b^*}^{r_j}, u_{\Pi(j)\cdot b^*}^{dk_i^* r_j})$ for a new random key $dk_i^* \leftarrow \mathbb{Z}_q$ which from then on plays the role of dk_i in determining whether participant $\mathcal{P}_i$ has won the election.

We refer to [15] for the proof that $\mathtt{H}_4[x]$ is indistinguishable from $\mathtt{H}_3[x]$ assuming that the DDH assumption holds in $\mathbb{G}$. We now show that for all adversaries, we have that:

$$\mathtt{Pr}\left[\mathtt{H}_4[x \quad | \quad i \in \{1,\ldots,N\}\backslash M]\right] \leq \frac{1}{n^*}$$

with $n^* \geq 2$ being the number of uncorrupted participants in the bucket b^*. In $\mathtt{H}_4[x]$, all the uncorrupted participants' entries in bucket b^* appear random, *i.e.* as (g^a, g^b) with a and b random values selected from $\mathbb{Z}_q$. Thus the contents of bucket b^* are distributed independently of the "winning" participant $\mathcal{P}_i$. Thus the adversary $\mathcal{A}$ can do no better than choosing an uncorrupted participant at random from the set of participants registered in bucket b^*. In the best case, the bucket b^* contains the minimum value of uncorrupted participants. Thus the adversary wins the unpredictability game in $\mathtt{H}_4[x]$ with probability at most $\frac{1}{n^*}$; that is, with probability $\frac{1}{2}$ in the best case.

5.4.4 Liveness: a strategy to break the property

Informally, the liveness property requires that even if a fraction of participants is controlled by an adversary $\mathcal{A}$, at least one participant provides a valid leader eligibility proof accepted by other participants at each election. We provide a strategy that shows that the liveness property is not guaranteed in the shuffling-based SSLE scheme.

By construction of the shuffling-based SSLE scheme, each participant secretly determines if he is elected as leader and his identity remains hidden until he decides to reveal his leader eligibility proof. Indeed, each participant determines with his position p_i, where his secret k_i is committed in $\mathcal{L}$, if this equality $p_i - z' = R \mod (N - z)$ is verified. If this is the case, the elected participant broadcasts his eligibility proof $\pi = (i, k_i, p_i)$. Thus, the non-elected participants cannot learn who is the leader before he reveals his eligibility proof π since we cannot learn which participant has committed the secret at the position p_i in $\mathcal{L}$ selected with Q without the knowledge of k_i. Moreover, at each election, exactly one participant is chosen as leader by the uniqueness property of the shuffling-based SSLE scheme.

The strategy is as follows. In the case where a corrupted participant $\mathcal{A}_t$ is chosen as leader, this means that $\mathcal{A}_t$ obtains the output $(1, \pi_t) \leftarrow \mathtt{SSLE.Elect}(t, \mathtt{pp}, \mathtt{st}, Q, sk_t, p_t)$. The corrupted leader $\mathcal{A}_t$ may choose to not reveal that she is elected and does not sent $(1, \pi_t)$ to other participants. By the uniqueness of the shuffling-based SSLE scheme, the other participants $\mathcal{P}_i$ output $(0, \perp) \leftarrow \mathtt{SSLE.Elect}(i, \mathtt{pp}, \mathtt{st}, Q, sk_i, p_i)$. Therefore, for this election there is no leader with a non-negligible probability of $\frac{f}{n}$.

5.5 Tweakened shuffling-based SSLE

In this section, we provide a modification of the shuffling-based SSLE scheme to detect the strategy of Section 5.4.2 by adding a checking step. Then, we provide a security analysis of this modified shuffling-based SSLE scheme, called *tweakened shuffling-based SSLE*.

5.5.1 Tweakened shuffling-based SSLE construction

When a participant $\mathcal{P}_i$ is elected, he has to reveal the value dk_i and k_i. With the knowledge of dk_i, anyone can detect that the coalition of f malicious participants has misbehaved. Indeed, once the proof of the elected participant is provided, anyone can parse the current state $\texttt{st} = (\mathcal{L}, pk_1, \ldots, pk_n)$ and run the following check. For all p_j such that $p_j \mod \sqrt{N} = p_i \mod \sqrt{N}$, the corresponding participant is in the same bucket as the elected participant $\mathcal{P}_i$. Then, for all p_j, anyone can check whether the corresponding entry $\mathcal{L}_j = (u_j, v_j)$ is such that $u_j^{dk_i} = v_j$. If yes, then there is a duplicate and the election has to be invalidated. Note that this additional check enables to detect not only that there is a duplicate but also the number of duplicates. Another election occurs to elect a leader by deleting the duplicated value and its duplicates in $\mathcal{L}$. We can be sure that the duplicated value and its duplicates are owned by corrupted participants since the algorithm $\texttt{SSLE.RegisterVerify}(i,\texttt{pp},\texttt{st},dk_i,p_i)$ ensures that no one can duplicate an entry of an uncorrupted participant without being detected. So, a participant who has duplicated an entry in $\mathcal{L}$ is a corrupted participant and thus, duplicated entries correspond to a registered corrupted participant.

In order to detect the strategy described in section 5.4.2, we propose a slight modification of the process $\texttt{SSLE.Verify}(i,\texttt{pp},\texttt{st},Q,\pi)$ as follows.

- $\texttt{SSLE.TVerify}(i,\texttt{pp},\texttt{st},Q,\pi)$: Parse $\texttt{st} = (\mathcal{L}, pk_1, \ldots, pk_n)$, $\pi = (i, p_i, k_i)$ and $\mathcal{L}_i = (u, v)$. Compute $(dk_i', pk_i') = H(k_i)$. If $p_{i-z'} = Q \mod (N - z)$, $u^{dk_i'} = v$ and $pk_i' = pk_i$, output 1. Otherwise, output 0.
  For all p_j such that $p_j \mod \sqrt{N} = p_i \mod \sqrt{N}$, then the corresponding participant is in the same bucket as the elected participant $\mathcal{P}_i$. Anyone can check whether the corresponding entry $\mathcal{L}_j = (u_j, v_j)$ is such that $u_j^{dk_i} = v_j$. If yes, then there is a duplicate. Then, replace all entries $\mathcal{L}_j = (u_j, v_j)$ such that $u_j^{dk_i} = v_j$ by $\perp$ and the output is 0. Otherwise, all the above checks pass and the output is 1.

5.5.2 Security analysis of tweakened shuffling-based SSLE

In this section, we analyze the tweakened shuffling-based SSLE scheme described in Section 5.5.1. In particular, we provide a strategy that breaks the liveness property

and we prove the following theorem.

Theorem 3 *Let $\mathbb{G}$ be a group in which DDH problem is hard. For any adversary $\mathcal{A}$, the tweakened shuffling-based SSLE construction is a unique, fair and $\left(\frac{1}{n*} + \texttt{negl}(\lambda)\right)$-unpredictable SSLE scheme in the random oracle model where n^* is the minimum number of uncorrupted participants in buckets containing uncorrupted participants.*

Proof. We provide the proofs for the properties of uniqueness, fairness and unpredictability that therefore prove Theorem 3.

Uniqueness proof: The `SSLE.RegisterVerify` and `SSLE.Elect` algorithms ensure that the element in $\mathcal{L}$ selected with the random value Q corresponds exactly to one secret k_i and so exactly one participant is elected. The new check step done in `SSLE.TVerify` does not impact neither `SSLE.RegisterVerify` nor `SSLE.Elect` and ensures also that the winner with the corresponding secret k_i has committed only once his secret in the list $\mathcal{L}$. Thus, the proof of the uniqueness is the same as Section 5.4.1 and proves that the tweakened shuffling-based SSLE guarantees the uniqueness property.

Fairness proof: The fairness property follows directly from the construction where exactly one entry is selected uniformly at random to be the leader in each election. The checks done in `RegisterVerify` ensure that each uncorrupted participant registers once in $\mathcal{L}$ with one secret dk_i and thus each entry in $\mathcal{L}$ corresponds to one uncorrupted participant. Thus, any given uncorrupted participant has a probability $\frac{1}{n}$ of being elected. The new checks done in `SSLE.TVerify` ensure that a corrupted participant whose secret is duplicated in several entries in $\mathcal{L}$ is discarded from the election and `SSLE.TVerify` outputs $\perp$ if this participant is selected as leader in `SSLE.Elect`. Then, every corrupted participant has to be registered once in $\mathcal{L}$ with her corresponding secret to have the chance to be leader and so has also a probability $\frac{1}{n}$ of being elected.

Unpredictability proof: The new check step done in `SSLE.TVerify` does not reveal any information on the elements in the bucket $b*$ in which the leader is selected. Thus, the elements in $b*$ appear random to the adversary $\mathcal{A}$ until the corresponding secret is revealed. If the adversary $\mathcal{A}$ does not control the leader, then the leader $\mathcal{A}$ can do not better than choosing an uncorrupted participant at random from $b*$ to predict the leader. Thus, the proof of the unpredictability is the same as Section 5.4.3 and proves that the tweakened shuffling-based SSLE scheme guarantees the unpredictability property.

Strategy against the liveness. In the tweakened shuffling-based SSLE scheme of Section 5.5.1, we can apply the same strategy described in Section 5.4.4. Indeed, as in the shuffling-based SSLE scheme, each participant secretly determines if he is elected as leader and remains hidden until he reveals his eligibility proof.

Thus, a corrupted leader can choose to not reveal her eligibility proof and by the uniqueness property of the tweakened shuffling-based SSLE scheme, there is no leader for this election. Therefore, the liveness property is not satisfied in the tweakened shuffling-based SSLE scheme. The detail of the strategy can be found in Section 5.4.4.

5.6 Discussion on the fairness property

In this section, we motivate our definition of the fairness property. Let us first recall both definitions of fairness:

- *Definition 1* [15]: each uncorrupted participant has the probability $\frac{1}{n}$ to be elected and in a set of f corrupted participants, the probability that one of them is elected leader is $\frac{f}{n}$.
- *Definition 2* : each participant, even in the set of corrupted participants, has the probability $\frac{1}{n}$ to be elected leader.

Note that in Definition 1, it is possible that in the set of f corrupted participants, the probability that one of them is elected leader is $\frac{f}{n}$ and the other $f-1$ corrupted participants have a probability zero of being elected. We compare these two definitions in order to outline that in some cases satisfying only Definition 1 may lead to an issue whereas satisfying Definition 2 prevents this issue. For that, we analyze the use of the leader election output to construct a random beacon [91] which is a common generator to obtain a random value Q. We consider two constructions: (1) a first construction using the result of the election process as an external source of entropy such as using the result of Bitcoin consensus protocol in [18]; and (2) a second construction using an internal source such as the history of a previous election like in Algorand [29] and Fantomette [7].

5.6.1 Random beacon construction from an external source

In [18], Bonneau *et al.* propose a construction of random beacons from the result of the Bitcoin consensus protocol and show the importance of manipulation-resistant property of random beacon that guarantees that the output Q cannot be manipulated to be more advantageous for the adversary.

Following this idea, a random beacon value Q could be computed with the result of the shuffling-based SSLE leader election, *e.g.* data of the leader such as his identity i and his secret k_i. If we use the SSLE scheme according to Definition 1 of Section 5.3, a set of f corrupted participants can run the strategy of Section 5.4.2 to bias the leader election and thus the random beacon construction.

To this end, they choose the most advantageous pair (i, k_i) that belongs to a corrupted participant $\mathcal{A}_i$ and duplicate k_i in f positions in $\mathcal{L}$. If $\mathcal{A}_i$ is elected as leader, then this means that one of f positions in $\mathcal{L}$ has been selected. The random value Q is then computed using the pair (i, k_i). Thereafter, Q could be used in applications requiring randomness such as a lottery or another random beacon for a consensus protocol.

With this random beacon construction that uses the result of the SSLE leader election according to Definition 1 as described in Section 5.3, the random value Q can be manipulated by a set of corrupted participants. In particular, the pair (i, k_i) is chosen such that the corresponding Q is advantageous for corrupted participants, *e.g.* being the winner of the lottery or being the leader that provides a block in a consensus protocol for blockchain, since the probability to select the secret k_i is $\frac{f}{n}$ compared to a uniform probability.

In the case where this random beacon is constructed using the SSLE leader election according to Definition 2 as described in Section 5.5, selecting a pair (i, k_i) cannot occur with probability better than $\frac{1}{n}$. Indeed, duplicates in $\mathcal{L}$ are detected and this ensures that a given secret is only committed in one position in $\mathcal{L}$.

5.6.2 Random beacon from an internal source

In the original SSLE paper [15], the authors propose that a fresh random value Q is generated for each election and the mean to generate it is out of the scope of their paper. Several works have studied how to generate this random beacon [18, 29, 7].

As in [29, 7], the random beacon construction can use data of the leader election protocol itself. Hereinafter, we provide a construction of the random value Q_r for an election r from the output of the shuffling-based SSLE leader election. Then, we include this construction in the shuffling-based SSLE leader election according to Definition 1 as described in Section 5.3 to outline the problems that may arise.

The random beacon uses a first random value Q_0 that may be chosen by all participants. Note that a random beacon construction that would be only based on public parameters of SSLE leader election is not suitable because it would be predictable. This would contradict the unpredictability property of SSLE leader election. Therefore, the construction of the random beacon needs to use the secret information such as the secret key k_i of the leader. Let's assume that the random value Q_r used in the election r to elect a leader l_r is computed as follows: $Q_r = HASH(l_{r-1}, k_{l_{r-1}})$ where l_{r-1} is the identity of the leader elected in the previous election $r-1$ and $k_{l_{r-1}}$ his secret key. Let us detail the shuffling-based SSLE leader election fulfilling fairness as defined in Definition 1 with this random beacon construction. The set of n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ run the election r to elect a leader as follows:

- The participants of the election r registered in $\mathcal{L}$ are those already registered in the previous election $r-1$ and also the new participants which register by running the algorithms of `SSLE.KeyGen` and `SSLE.Register` such that the output of `SSLE.RegisterVerify` algorithm is 1. Note that the leader of the round $r-2$ and the new participants have to register before the end of round $r-1$ if they want to participate in the election r.

- After obtaining $\mathcal{L}$, each participant $\mathcal{P}_i$ computes the random Q_r using the pair $(l_{r-1}, k_{l_{r-1}})$ resulting from the previous election: $Q_r = HASH(l_{r-1}, k_{l_{r-1}})$

- Each participant $\mathcal{P}_i$ runs `SSLE.Elect` with Q_r as random value. If `SSLE.Elect` outputs 1, then $\mathcal{P}_i$ knows that he is elected as the leader of the election r. The participant $\mathcal{P}_i$ provides his eligibility proof $\pi = (i, p_i, k_i)$ to other participants.

- Each participant $\mathcal{P}_j$ verifies the leader eligibility proof π of $\mathcal{P}_i$ with `SSLE.Verify`. If the output is 1, then $\mathcal{P}_j$ considers $\mathcal{P}_i$ as the leader of the election r. The random value Q_{r+1} will be computed with his pair (i, k_i): $Q_{r+1} = HASH(i, k_i)$.

Note that the random value Q_{r+1} will be used in the election $r+1$ to elect the next leader. Moreover, the registration process for an election $r+1$ has to be completed before the leader of the election r reveals his identity. This ensures that the random value Q_{r+1} is only computed once all participants have committed their secrets. The leader of the election r cannot register for the election $r+1$ since he knows the index in the list which is selected by Q_{r+1}.

Since this SSLE leader election only fulfills the fairness property according to Definition 1, a set of t corrupted participants can run the strategy described in Section 5.4.2 in order to bias the leader election and so the random beacon construction, in the election r for instance. To this end, $f < t$ corrupted participants choose a pair (i, k_i) that belongs to the corrupted participant $\mathcal{A}_i$ and duplicate k_i in f positions in $\mathcal{L}$. In particular, they choose (i, k_i) such that the random $Q_{r+1} = HASH(i, k_i)$ would select a position in $\mathcal{L}$ where one of the $t-f$ other corrupted participants are registered. The random value Q_{r+1} is advantageous for the corrupted participants since the probability to select the secret k_i is $\frac{f}{n}$ instead of the uniform probability. Note that the $f-1$ duplicates of k_i in $\mathcal{L}$ become useless if k_i is selected since a pair of public-secret keys $(pk_i, (k_i, dk_i))$ is one-time use for the eligibility proof.

If the shuffling-based SSLE leader election fulfills the fairness property according to Definition 2, selecting a pair (i, k_i) cannot occur with probability better than $\frac{1}{n}$ since any duplication of a secret value in the list of participants is detected.

Through these two random beacon constructions, we show that the manipulation-resistant property of random beacon can depend on the considered definition of fairness.

5.7 Summary on SSLE

In this chapter, we revisited the single secret leader election protocol based on the hardness of DDH. Since this protocol originally described in [15] is the most practical one among the three proposed constructions, we considered that it deserves further study. We first added the liveness property in the security model of a SSLE scheme to ensure a leader election even in presence of malicious or inactive participants. Then, we revisited the fairness property to ensure a uniform probability of being elected even among corrupted participants. We refined also the unpredictability property to ensure that there is at least two uncorrupted nodes in the list from which the leader is elected and prevent the adversary learning which participant has been elected.

Then, we considered the case of a collaborative set of participants that aim to introduce a bias so as to make one of them elected with probability $\frac{f}{n}$ where f is the number of corrupted participants compared to $\frac{1}{n}$. Note that, for a specific election, the other $f-1$ corrupted participants have a probability zero of being elected. We then explained how to detect this attack strategy and we provided a slight modification of the specification of SSLE based on the hardness of DDH by adding a checking step to be performed after the leader has been revealed.

Finally, this work aims to outline the importance of the fairness property where all participants, corrupted and uncorrupted, have the same probability to be elected as leader. Thus, we also provided two constructions of random beacon to motivate the need for our definition of the fairness property.

Note that, the fairness definition considers equal weighting of participants that can easily be extended to consider participants with respect to some other distribution. For example, if the shuffling-based SSLE scheme is combined with a PoS mechanism, each unit of money can be considered as one registration.

Chapter 6

Unpredictability properties in Algorand

Contents

6.1 Algorand overview

A promising approach to construct consensus protocols for blockchain is the Algorand consensus protocol [28, 53, 29] used in the blockchain of the same name. Algorand consensus protocol can be considered as a Probabilistic Leader Election (PLE) protocol where one leader is elected on expectation [6]. An interesting mechanism of Algorand is a new Byzantine agreement (BA) protocol based on a *Verifiable Random Function* (VRF). A BA is a protocol enabling to reach a common agreement on a value assuming Byzantine faults. A VRF [80] is a function that takes as inputs a pair of message and secret, and outputs a random value Q and a proof π that Q is correctly computed.

Algorand consensus protocol is divided in rounds. At the first round $k = 0$, a first election randomness value Q_0 is agreed by the initial participants. Then, several rounds occurs. At each round, a leader election protocol is run to elect a participant as leader. First, each participant secretly determines if he is selected as potential leader by computing the VRF function with as inputs his private key and the public randomness value. His identity remains hidden until he decides to reveal his leader eligibility proof. A number of potential leaders n_l is expected at each election. When the identities of potential leaders are revealed, a rule enables to choose one of them as leader. Then, several steps occur to acknowledge the chosen leader. Finally, a new random value is computed with the leader eligibility proof and is intended to be used as random value for the next election. In the case where there is no leader, the round is *empty* and the random value is computed with the default computation.

In the original papers of Algorand [28, 53, 29], Chen and Micali introduce the Algorand blockchain and consensus protocols. Algorand consensus protocol was designed to operate while more than 2/3 of money are honest, *i.e.* more than 2/3 of money belongs to honest participants. They show that with overwhelming probability, (a) all honest participants agree on the same block B_k generated at round k and (b) the leader of a new block is honest with a probability of at least $p_h = h^2(1 + h - h^2)$ with h the percentage of participants who are honest. In [53], Gilad *et al.* apply the Algorand consensus protocol [28] to achieve safety and liveness in partial synchrony in the context of cryptocurrency

Although Algorand delivers promising result, the papers [100, 32] argue that its security assumptions cannot be always guaranteed. In [100], Wang provides two attacks intended to show that the assumptions of fork occurs with a small probability and the honest majority of money may not be guaranteed if the attacker bribes participants. In [32], Conti *et al.* present a DDoS attack where the adversary floods a participant of *undecidable* messages in order to make him unavailable or isolate him from other participants.

These different analysis of Algorand [28, 29, 53, 100, 32] have not studied the *unpredictability* property as defined in [15] that we study in this chapter. Note that, we consider Algorand in term of number of participants that can be easily adapted in term of money.

6.2 Security model

Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be the n initial participants who participate in the election process. In a Probabilistic Leader Election (PLE) protocol, one leader is elected on expectation [6] at each round $k \geq 1$.

6.2.1 PLE Model

Let λ be the security parameter. We adapt the formal model of a SSLE [15] scheme to fit better with PLE protocols such as Algorand. It is defined as a tuple of Probabilistic Polynomial Time (PPT) algorithms and protocol $\texttt{PLE} = (\texttt{PLE.Setup}, \texttt{PLE.KeyGen}, \texttt{PLE.Register}, \texttt{PLE.RegisterVerify}, \texttt{PLE.Elect}, \texttt{PLE.Verify})$ defined as follows.

The algorithm `PLE.Setup` is a process intended to be run a single time at the beginning of the protocol:

- $\texttt{PLE.Setup}(1^\lambda, \texttt{param}) \rightarrow \texttt{pp}, \texttt{stpub}_0$. On input 1^λ and the set of parameters `param`, it generates the public parameters `pp` and the initial state $\texttt{stpub}_0$.

The algorithm `PLE.KeyGen` is run by each participant before the registration process to generate the keys to participate in the protocol:

- $\texttt{PLE.KeyGen}\ (1^\lambda, \texttt{pp}) \rightarrow (sk_1, pk_1), (sk_2, pk_2)$. On input 1^λ and the public parameters pp, it generates two pairs of secret and public keys (sk_1, pk_1) used for the election and (sk_2, pk_2) used for the registration and signing processes.

The algorithm `PLE.Register` is run by an already registered participant to register a new participant to take part in the consensus protocol:

- $\texttt{PLE.Register}(i, \texttt{pp}, k, \texttt{stpub}_k, (sk_{i,1}, pk_{i,1}), (sk_{i,2}, pk_{i,2}), (pk_{i',1}, pk_{i',2})) \rightarrow \texttt{stpub}_{k+1}, \texttt{stpriv}_i$. An already registered participant $\mathcal{P}_i$ registers a new participant $\mathcal{P}_{i'}$ to take part in the elections by using the public parameter `pp`, the round index k, the corresponding state $\texttt{stpub}_k$, the secret and public keys $(sk_{i,1}, pk_{i,1}), (sk_{i,2}, pk_{i,2})$ and the two public keys $(pk_{i',1}, pk_{i',2})$. It computes a registration value rv with $(sk_{i,2}, pk_{i,2})$ and $(pk_{i',1}, pk_{i',2})$. It modifies the current state $\texttt{stpub}_k$ to $\texttt{stpub}_{k+1}$ by adding $\{rv, pk_{i',1}, pk_{i',2}\}$ to $\texttt{stpub}_k$. Registration outputs the updated state $\texttt{stpub}_{k+1}$ and $\texttt{stpriv}_i = (sk_{i,1}, sk_{i,2})$.

Every time a new participant has registered, previously registered participants have to run the `PLE.RegisterVerify` algorithm to verify that the registration has been correctly carried out:

- $\texttt{PLE.RegisterVerify}(i, \texttt{pp}, k, \texttt{stpub}_k) \rightarrow 0/1$. An already registered partici-
-pant $\mathcal{P}_i$ verifies the registration of a new participant by using the public parameters `pp`, the index of the round k and the corresponding public state $\texttt{stpub}_k$. If the state $\texttt{stpub}_k$ is valid, it outputs 1. Otherwise, it outputs 0.

At each round, the `PLE.Elect` protocol is run by each participant to elect a leader for this round. During the `PLE.Elect` process, several steps occur and the participants interact to agree on a leader:

- PLE.Elect(i,pp,k,stpub$_k$,stpriv$_i$,Q_{k-1}) $\rightarrow$ $(0,\perp)$/ $(1,\pi)$. It takes as inputs the public parameter pp, the index of the round k, the state stpub$_k$, the private state stpriv$_i$ and the election randomness value Q_{k-1}. In the first step, the participant $\mathcal{P}_i$ verifies his own leader eligibility and broadcasts his leader eligibility proof in case he is elected. Then, several steps occur to collect a threshold of vote messages for a leader proof or for an empty round. It outputs $(1,\pi)$ with $\pi = \pi_{i'}$ if there is a participant $\mathcal{P}_{i'}$ elected as the leader for this round k or $\pi = \pi_{k,\varepsilon}$ the empty round proof if there is no leader for this round k. In both cases, a new value Q_k is also computed. Otherwise, it outputs $(0,\perp)$.

The process PLE.Verify is used to check whether the participant selected as leader has been correctly elected or the empty round has been correctly voted:

- PLE.Verify(i,pp,k,stpub$_k$,Q_{k-1},π)$\rightarrow 0/1$. Given the public parameter pp, the index of the round k, the corresponding public state stpub$_k$, the election randomness value Q_{k-1}, the proofs π claiming that a particular participant was elected leader or an empty round was acknowledged, the verification algorithm accepts or rejects the proof that the participant has been elected leader or an empty round was acknowledged.

6.2.2 Security properties

In this chapter, we focus on the unpredictability aspect of Algorand. Therefore, in this section, we formally define two unpredictability properties. In particular, we adapt the unpredictability game provided in Chapter 5 and extend this unpredictabi--lity property to the new property named t-forward unpredictability. First, we provide a generic experiment that applies for the two defined security properties. Then, we provide the challenge phase specifically defined for each property.

Generic experiment

It is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **PLE.Setup phase.** The adversary $\mathcal{A}$ selects the parameter param. The challenger $\mathcal{C}$ runs PLE.Setup(1^λ, param) and gives the parameters pp and stpub$_0$ to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ selects a number f of participants to corrupt and a related subset M of size f. The challenger $\mathcal{C}$ runs the algorithm of PLE.KeyGen(1^λ,pp) for the uncorrupted participants and gets corresponding values $(sk_{i,1}, pk_{i,1})$ and $(sk_{i,2}, pk_{i,2})$.

- **Election Phase.** The challenger $\mathcal{C}$ generates the first election randomness Q_0. Then, the corrupted and uncorrupted participants are registered. In either

case, the challenger $\mathcal{C}$ verifies the registration with `PLE.RegisterVerify(`i`,pp,`k`,`$\texttt{stpub}_k$`)`. If it returns 0, the game immediately ends with output 0. Otherwise, the state is updated to $\texttt{stpub}_{k+1}$. We denote by n the number of registered participants. During this phase, several elections may occur. For each new election, the challenger $\mathcal{C}$ runs `PLE.Elect(`i`,pp,`k`,`$\texttt{stpub}_k$`,`$\texttt{stpriv}_i$`,` Q_{k-1}`)` on behalf of each uncorrupted participant $\mathcal{P}_i$ with the random value Q_{k-1} computed with the result of the election at the previous round $k-1$. Then, $\mathcal{C}$ sends the outputs $(1, \pi)$ or $(0, \perp)$ to the adversary $\mathcal{A}$.

After the election phase, a challenge phase is executed. This latter phase is specifically defined for each property.

Unpredictability property

The unpredictability property defined in [15] means that an adversary $\mathcal{A}$ who does not control the leader cannot learn which participant has been elected. As discussed in Chapter 5, this property originally defined in [15] does not consider the case where there is exactly one uncorrupted participant into the list from which the winner of the election is elected, then $\mathcal{A}$ can predict with probability 1 who is the winner of the election. We recall the unpredictability game described in Chapter 5 that prevents this specific case.

The unpredictability experiment is denoted by $\texttt{UNPRED}[\mathcal{A}, \lambda, n, f]$ and is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge phase.** At some point, all participants have registered. Then, an election occurs. In this election, the challenger $\mathcal{C}$ does not send π_j for each uncorrupted participant $\mathcal{P}_j$ to $\mathcal{A}$. Let $\mathcal{P}_i$ be the winner of this election. The game ends with the adversary $\mathcal{A}$ outputting an index i'. If $\mathcal{P}_i$ is a corrupted participant or if $\mathcal{P}_i$ is the single uncorrupted participant registered in the list from which the leader is elected, then the experiment outputs 0. It outputs 1 if and only if $i = i'$. Else, it outputs 0.

We say that a PLE scheme is *unpredictable* if no PPT adversary $\mathcal{A}$ can win the unpredictability game with greater than negligible advantage when the winner of the election is uncorrupted. That is, for all PPT $\mathcal{A}$, we have:

$$Pr[\texttt{UNPRED}[\mathcal{A}, \lambda, n, f] = 1 | i \in \{1, \ldots, n\} \backslash M] \leq \frac{1}{n-f} + negl(\lambda)$$

t-forward unpredictability property

The definition of the unpredictability property only considers the current election and the case where the adversary does not control the winner of this election.

However, in some election such as Algorand, a corrupted leader can have an impact on the following elections as we show in Section 6.4.2. Moreover, in a context of blockchain, an attacker who knows at which election she will be leader can plan the data to add at the suitable time into the ledger. Therefore, we generalize the unpredictability property by introducing a new property called *t-forward unpredictability* to capture the case where even elected as leader, the adversary cannot predict the next leaders.

The t-forward unpredictability experiment is denoted by $\texttt{FUNPRED}[\mathcal{A}, \lambda, n, f]$ and is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge phase.** At some point, all participants have registered. Then, an election occurs. In this election, the challenger $\mathcal{C}$ does not send π_j for each uncorrupted participant $\mathcal{P}_j$ to $\mathcal{A}$. Let $\mathcal{P}_{i_0}$ be the winner of this election and $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_t}$ be the winners of the t following elections. The game ends with the adversary $\mathcal{A}$ outputting indexes $i'_0, i'_1, \ldots, i'_t$. It outputs 1 if and only if $i_0 = i'_0, i_1 = i'_1, \ldots, i_t = i'_t$. Else, it outputs 0.

We say that a PLE scheme is *t-forward unpredictable* if no PPT adversary $\mathcal{A}$ can win the forward unpredictability game with greater than negligible advantage. For all PPT $\mathcal{A}$, we have:

$$Pr[\texttt{FUNPRED}[\mathcal{A}, \lambda, n, f] = 1] \leq (\frac{f}{n} + (1 - \frac{f}{n})\frac{1}{n - f})\frac{1}{n^t} + negl(\lambda)$$

6.3 Algorand construction

In this section, we first recall the definition of verifiable random functions (VRF) and then provide the construction of Algorand.

Verifiable random function. A verifiable random function VRF [80] consists of three algorithms (G, E, V). Given a security parameter λ, G produces a pair of public and secret keys (pk, sk). The algorithm E takes as inputs a message m and a secret sk to produce the value v and the corresponding proof π. The algorithm V takes as inputs a message m, a public key pk, a value v and a proof π and accepts by outputting YES or refuses with the output NO. The properties [80] of a VRF function are the following:

- **Uniqueness**: no value $(pk, m, v, v', \pi, \pi')$ such that $v \neq v'$ can satisfy $V(pk, m, v, \pi) = V(pk, m, v', \pi') = YES$
- **Provability**: for all m, if $E(m, sk) = (v, \pi)$ then $V(pk, m, v, \pi) = YES$
- **Pseudorandomness**: v is indistinguishable from a truly random value.

Algorand [28, 53, 29] selects at each round/election k, one leader among n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$. At each election, a number of potential leaders n_l is expected to reveal their identities. When the identities of potential leaders are revealed, a rule enables to choose one of them as leader. Then, several steps occur to validate the participant selected as leader.

We denote by $VS_{k,s}$ the set of validators of round k at step s. Let $HASH$ be a hash function, $SIGN$ be a signature scheme and VRF be a verifiable random function.

- `PLE.Setup(`1^λ`,param)`: parse the set of parameters `param`$= \{n_l, n_v, lb, m\}$ where n_l is the expected number of potential leaders in each election, n_v is the expected number of validators in each step of an election, lb is the look back parameter to select the set of participants for an election and m the maximal number of steps in an election. Initialize $\mathcal{L} = \{\}$, $PK_0 = \{\}$ and the randomness Q_0. Output `pp`$= (n_l, n_v, lb, m)$ and $\texttt{stpub}_0 = (\mathcal{L}, PK_0, Q_0)$.

- `PLE.KeyGen (`1^λ`,pp)`: generate the secret and public keys $(sk_{i,1}, pk_{i,1})$ of the VRF function VRF; and the public and secret keys $(sk_{i,2}, pk_{i,2})$ of the signature scheme $SIGN$. Output $(sk_{i,1}, pk_{i,1}), (sk_{i,2}, pk_{i,2})$.

- `PLE.Register(`i`,pp,`k`,`$\texttt{stpub}_k, (sk_{i,1}, pk_{i,1}), (sk_{i,2}, pk_{i,2}), (pk_{i',1}, pk_{i',2})$`)`: parse the state $\texttt{stpub}_k$ and $PK_k = ((pk_{1,1}, pk_{1,2}, a_1), (pk_{2,1}, pk_{2,2}, a_2), \ldots, (pk_{i'-1,1}, pk_{i'-1,2}, a_{i'-1}))$. If $(pk_{i',1}, pk_{i',2}) \notin PK_k$, then $\mathcal{P}_i$ registers a new participant $\mathcal{P}_{i'}$ by generating the value $rv_{i,i'} = (pk_{i,1}, (pk_{i',1}, pk_{i',2}), a_{i'}, sign_i)$ where $sign_i = SIGN[sk_{i,2}](pk_{i,1}, (pk_{i',1}, pk_{i',2}), a_{i'})$. In other words, the value $rv_{i,i'}$ transfers an amount of money $a_{i'}$ from the account related to $pk_{i,1}$ to the new one related to $pk_{i',1}$. Update $\texttt{stpub}_k$ to $\texttt{stpub}_{k+1}$ by replacing $(pk_{i,1}, pk_{i,2}, a_i)$ by $(pk_{i,1}, pk_{i,2}, a_i - a_{i'})$ and by adding $(pk_{i',1}, pk_{i',2}, a_{i'})$ to PK_k and $rv_{i,i'}$ to $\mathcal{L}$. Output $\texttt{stpub}_{k+1}$ and $\texttt{stpriv}_i = (sk_{i,1}, sk_{i,2})$.

- `PLE.RegisterVerify(`i`,pp,`k`,`$\texttt{stpub}_k$`)`: parse the state $\texttt{stpub}_k = (\mathcal{L}, PK_k)$ and recover the value $rv_{j,j'} = (pk_{j,1}, (pk_{j',1}, pk_{j',2}), a_{j'}, sign_j)$ of the newly registered participant $\mathcal{P}_{j'}$. Verify $a_{j'} \leq a_j$ for the tuple $(pk_{j,1}, pk_{j,2}, a_j) \in PK_k$ and $sign_j$ with $pk_{j,2}$ is valid. If the above checks pass, then it outputs 1. Else it outputs 0.

- `PLE.Elect(`i`,pp,`k`,`$\texttt{stpub}_{k-lb}, \texttt{stpriv}_i, Q_{k-1}$`)`: The participant $\mathcal{P}_i$ such that $(pk_{i,1}, pk_{i,2}, a_i) \in \texttt{stpub}_{k-lb}$ runs the following steps during the round k.

  - At step $s = 1$, the participant $\mathcal{P}_i$ computes the target $pl_k = \frac{n_l}{|PK_{k-lb}|}$. Then $\mathcal{P}_i$ computes the VRF output $\sigma_{i,k} = VRF[sk_{i,1}](HASH(Q_{k-1}, k))$ and the corresponding VRF proof c_i.

* If $HASH(\sigma_{i,r}) \leq pl_k$, the participant $\mathcal{P}_i$ computes the contribution $\alpha_{i,k} = HASH(Q_{k-1})^{sk_{i,1}}$ and broadcasts his eligibility proof $\pi_i = (\sigma_{i,k}, c_i, \alpha_{i,k})$.

Then, in either case, $\mathcal{P}_i$ starts the step 2.

– At step $s = 2$, the participant $\mathcal{P}_i$ collects the proofs of potential leaders $\pi_{j'}$. Then, $\mathcal{P}_i$ computes the target $pv_k = \frac{n_v}{|PK_{k-lb}|}$ and the value $\gamma_{i,k,2} = VRF[sk_{i,1}](HASH(Q_{k-1}, k, 2))$. If $HASH(\gamma_{i,k,2}) \leq pv_k$, then $\mathcal{P}_i \in VS_{k,2}$ and he can select one proof $\pi_j = (\sigma_{j,k}, c_j, \alpha_{j,k})$ such that

* $HASH(\sigma_{j,k}) \leq pl_k$,
* $HASH(\sigma_{j,k}) \leq HASH(\sigma_{j',k})$ for all received VRF output $\sigma_{j',k}$
* and c_i is valid

Then, $\mathcal{P}_i$ sets $vote = \pi_j$ and broadcasts the vote message $vm_{i,2} = (\gamma_{i,k,2}, vote, sign_i)$ where $sign_i = SIGN[sk_{i,2}](vote)$. Else, if none proof passes the above verification steps or if $\mathcal{P}_i$ receives zero proof of potential leaders, then he broadcasts a vote message for the value $vote = \varepsilon$ that represents an empty round. Then, in either case, $\mathcal{P}_i$ starts the step 3.

– At step $s = 3$, the participant $\mathcal{P}_i$ collects the vote messages $vm_{j,s-1} = (\gamma_{j,k,s-1}, vote, sign_j)$ from validators in $VS_{k,2}$. If $\mathcal{P}_i \in VS_{k,3}$ that means that $HASH(\gamma_{i,k,3}) \leq pv_k$, and if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3} + 1$ of valid vote messages $vm_{j,2}$ for the same value $vote$ from different validators $\mathcal{P}_j$, then $\mathcal{P}_i$ broadcasts a vote message $vm_{i,3} = (\gamma_{i,k,3}, vote, sign_i)$ where $sign_i = SIGN[sk_{i,2}](vote)$. Else, $\mathcal{P}_i$ broadcasts a vote message for the value $vote = \varepsilon$. Then, in either case, $\mathcal{P}_i$ starts the step 4.

– At step $s = 4$, the participant $\mathcal{P}_i$ collects the vote messages $vm_{j,s-1} = (\gamma_{j,k,s-1}, vote, sign_j)$ from validators in $VS_{k,3}$. If $\mathcal{P}_i \in VS_{k,4}$ that means that $HASH(\gamma_{i,k,4}) \leq pv_k$, and if he receives at least $\frac{2n_v}{3} + 1$ of valid vote messages $vm_{j,3}$ for the same value $vote \neq \varepsilon$ from different validators $\mathcal{P}_j$, then $\mathcal{P}_i$ broadcasts a certify message $cm_{i,4} = (0, \gamma_{i,k,4}, vote, sign_i)$ where $sign_i = SIGN[sk_{i,2}](0, vote)$. Else, $\mathcal{P}_i$ broadcasts a certify message $cm_{i,4} = (1, \gamma_{i,k,4}, vote, sign_i)$ with $sign_i = SIGN[sk_{i,2}](1, vote)$ and the value $vote$ is defined as follows:

* $vote = \varepsilon$ if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3} + 1$ of valid vote messages $vm_{j,s-1} = (\gamma_{j,k,s-1}, vote, sign_j)$ for the same value $vote = \varepsilon$ from different validators $\mathcal{P}_j$.
* any $vote \neq \varepsilon$ if $\mathcal{P}_i$ receives at least $\frac{n_v}{3} + 1$ of valid vote messages $vm_{j,s-1} = (\gamma_{j,k,s-1}, vote, sign_j)$ for the same value $vote \neq \varepsilon$ from different validators $\mathcal{P}_j$.
* $vote = \varepsilon$ otherwise.

Then, in either case, $\mathcal{P}_i$ starts the step 5.

– At the step $5 \leq s \leq m-1$, if $\mathcal{P}_i \in VS_{k,s}$, *i.e.* $HASH(\gamma_{i,k,s}) \leq pv_k$, then he acts as follows:

  * *Ending Condition 0:* if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3}+1$ valid certify messages $cm_{j,s-1} = (0, \gamma_{j,k,s-1}, vote, sign_j)$ for the same value $vote = \pi_{j'}$ from different validators $\mathcal{P}_j$, then $\mathcal{P}_i$ computes $Q_k = HASH(\alpha_{j',k}, k)$, sets $\pi = (k, \pi_{j'}, CERT_{j'})$ where $CERT_{j'}$ is at least $\frac{2n_v}{3}+1$ valid certify messages $cm_{j,s-1}$ for $\pi_{j'}$ from different validators $\mathcal{P}_j$. The output is $(1, \pi)$.
  * *Ending Condition 1:* else, if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3}+1$ valid certify messages $cm_{j,s-1} = (1, \gamma_{j,k,s-1}, vote_j, sign_j)$ for any value $vote_j$, then $\mathcal{P}_i$ computes $Q_k = HASH(Q_{k-1}, k)$ and sets $\pi_\varepsilon = (k, \varepsilon, CERT_\varepsilon)$ where $CERT_{j\varepsilon}$ is at least $\frac{2n_v}{3}+1$ valid certify messages $cm_{j,s-1}$ for any value $vote_j$ from different validators $\mathcal{P}_j$. The output is $(1, \pi_\varepsilon)$.
  * else, $\mathcal{P}_i$ recovers the value $vote$ as in step $s = 4$ and broadcasts $cm_{i,s} = (1, \gamma_{i,k,s}, vote, sign_i)$ if $\mathcal{P}_i$ receives more than $\frac{2n_v}{3}+1$ valid certify messages of the form $cm_{j,s-1} = (1, \gamma_{j,k,s-1}, vote_j, sign_j)$ for any value $vote_j$. Else, $\mathcal{P}_i$ broadcasts $cm_{i,s} = (0, \gamma_{i,k,s}, vote, sign_i)$. Then, in either case, $\mathcal{P}_i$ starts the step $s+1$.

– else, this means $s = m$ *, $\mathcal{P}_i$ verifies *Ending Condition 0.* Else, he verifies *Ending Condition 1.* Else $\mathcal{P}_i$ sets $vote = \varepsilon$ and broadcasts $cm_{i,m} = (1, \sigma_{i,k,m}, vote, sign_i)$.

Else, this means $HASH(\sigma_{i,k,s}) > pv_r$ or the step m does not enable to output. At any step $s > 4$ if $\mathcal{P}_i$ enters in *Ending condition 0* or *Ending condition 1.* Else the output is $(0, \perp)$.

- `PLE.Verify(`i`,pp,`k`,stpub`$_{k-lb}, Q_{k-1}, \pi$`)`: If $\pi \neq \pi_\varepsilon$, parses $\texttt{stpub}_{k-lb}$ and $\pi = (k, (\sigma_{j,k}, c_j, \alpha_{j,k}), CERT_j)$. Verify if c_j is valid, $HASH(\sigma_{j,k}) \leq pl_k$ and $HASH(\sigma_{j,k}) \leq HASH(\sigma_{j',k})$ for all received VRF outputs $\sigma_{j',k}$. If $CERT_j$ is a set of at least $\frac{2n_v}{3}+1$ valid certify messages for π from different validators and for $s \geq 4$, then $\mathcal{P}_i$ computes $Q_k = HASH(\alpha_{j,k}, k)$ and output 1. Else, if $\pi = \pi_\varepsilon$, the participant $\mathcal{P}_i$ verifies that π is a set of at least $\frac{2n_v}{3}$ valid certify messages for any value. If yes, he computes $Q_k = HASH(Q_{k-1}, k)$ and output 1. Otherwise, output 0.

The next election is then run at the round $k+1$ with the public state $\texttt{stpub}_{k+1-lb}$ and the random value Q_k.

*m is chosen such that with overwhelming probability, the `PLE.Elect` protocol has ended before this step m [28, 29]

6.4 Security analysis

In this section, we analyze the unpredictability and t-forward unpredictability properties defined at Section 6.2.2.

6.4.1 Unpredictability proof

The unpredictability property means that an adversary $\mathcal{A}$ who does not control the leader cannot learn which participant has been elected. Let $HASH$ be a random oracle. In this section, we provide a proof of the following theorem:

Theorem 4 *For any adversary $\mathcal{A}$, the Algorand leader election protocol provided in Section 6.3 is unpredictable in the random oracle model under the assumption of the pseudorandomness property of the verifiable random function.*

Proof. Suppose that the adversary $\mathcal{A}$ wins the unpredictability game. Then, at some point of the challenged election k, the adversary $\mathcal{A}$ outputs the index i' such as $\mathcal{P}_i$ is an uncorrupted participant and $\texttt{PLE.Verify}(i,\texttt{pp},k,\texttt{stpub}_{k-lb}, Q_{k-1}, \pi)$ with $\pi = (k, \pi_{i'}, CERT_{i'})$ outputs 1. This means that the adversary $\mathcal{A}$ can predict which VRF outputs $\sigma_{i',k}$ is the lowest value such that $HASH(\sigma_{i',k}) \leq pl_k$. Then, this means that the adversary $\mathcal{A}$ can distinguish $\sigma_{i',k}$ from a random value with a non-negligible advantage. This implies that the attacker has broken the pseudorandomness property of the VRF function.

Therefore, the adversary cannot do better than randomly selecting an uncorrupted participant in the list $\mathcal{L}_k$ of registered participants to know who has been elected at election k, *i.e.* with a probability of at most $\frac{1}{n-f}$.

6.4.2 Strategy against the t-forward unpredictability

In this section, we provide a strategy that breaks the t-forward unpredictability property for the specific setting $n_l = 1$. This means that if the leader does not reveal her leader eligibility proof, then there is no leader for this election and the next random value is computed with the default computation. The strategy enables an adversary $\mathcal{A}$ elected at a given election to predict the leaders of the t next elections.

The idea of the attack is the following. At the election of a round k, a corrupted participant is elected as leader and sends her eligibility proof π only to the corrupted participants. If the next random value computed with the proof π elects corrupted participants as leaders for the next elections, then π is revealed to the uncorrupted participants. Else, if the next random value computed with the default computation enables corrupted participants being elected as leaders for the next elections, then

π is not revealed to the uncorrupted participants. By default, π is revealed to the uncorrupted participants. Thus, with a non-negligible advantage, the corrupted participants can predict the leaders of the next elections with this strategy. First, we describe the strategy in more detail in the next paragraph for one following election and then we extend the strategy for any number t of following elections.

Strategy for one following election

Let $\mathcal{A}$ be a set of f corrupted participants $\mathcal{A}_1, \ldots, \mathcal{A}_f$. We assume that at the election of a round k, a corrupted participant $\mathcal{A}_l$ is elected as leader and generates her leader eligibility proof $\pi_l = (\sigma_{l,k}, c_i, \alpha_{l,k})$ as computed in Section 6.3. Then, $\mathcal{A}_l$ sends π_l only to $\mathcal{A}$ and each $\mathcal{A}_i$ performs the following protocol during the step 1 of `PLE.Elect`:

Each $\mathcal{A}_i$ computes the following values

- The two possible next random values $Q_k = HASH(\alpha_{l,k}, k)$ and $Q_{k,\varepsilon} = HASH(Q_{k-1}, k)$.
- The two possible VRF outputs $\sigma_{i,k+1} = VRF[sk_{i,1}](HASH(Q_k, k+1))$ and $\sigma'_{i,k+1} = VRF[sk_{i,1}](HASH(Q_{r,\varepsilon}, k+1))$.

Then, the set of corrupted participants checks the following verification steps:

- If $HASH(\sigma_{i,k+1}) \leq pl_{k+1}$ (Eq1) for any corrupted participant $\mathcal{A}_i$, then $\mathcal{A}_l$ reveals π_l to the uncorrupted participants
- else, if $HASH(\sigma'_{i,k+1}) \leq pl_{k+1}$ (Eq2) for any corrupted participant $\mathcal{A}_i$, then $\mathcal{A}_l$ does not reveal π_l.
- else, this means that none corrupted participant is elected as the leader at round $k+1$ neither with $\sigma_{i,k+1}$ nor with $\sigma'_{i,k+1}$, then $\mathcal{A}_l$ reveals π_l to the uncorrupted participants

Thereafter, the other steps of `PLE.Elect` protocol are performed in regular manner.

Thus, the probability that one of corrupted participants is elected at round k and the adversary $\mathcal{A}$ can predict the leader at round $k+1$ is $\frac{f}{n}(\frac{f}{n} + (1 - \frac{f}{n})\frac{f}{n})$ instead of $\frac{f}{n} \cdot \frac{1}{n}$. Indeed, the leader at round k is a corrupted participant with probability $\frac{f}{n}$. A leader is corrupted at round $k+1$ with Q_k, *i.e.* Eq1 is verified, with probability $\frac{f}{n}$. A corrupted participant is elected at round $k+1$ with $Q_{r,\varepsilon}$, *i.e.* participants check Eq2 if Eq1 is not verified with probability $(1 - \frac{f}{n})$, and Eq2 is verified with probability $\frac{f}{n}$. The probability that one of uncorrupted participants is elected as leader at round k and $\mathcal{A}$ can predict the leader at round $k+1$ does not change with the selfish strategy and is $(1 - \frac{f}{n})\frac{1}{n-f} \cdot \frac{1}{n}$. Finally, the adversary can win the 1-forward predictability game with probability $\frac{f}{n}(\frac{f}{n} + (1 - \frac{f}{n})\frac{f}{n}) + (1 - \frac{f}{n})\frac{1}{n-f} \cdot \frac{1}{n}$.

Strategy for t following elections

The strategy described in the previous paragraph can be extended for any t following elections. Indeed, with the two possible randomness values $Q_k = HASH(\alpha_{l,k}, k)$ and $Q_{k,\varepsilon} = HASH(Q_{k-1}, k)$, the corrupted participants verify which random value enables to consecutively elect corrupted leaders until the round $k+t$. If this is the case with Q_k, then π_l is revealed to the uncorrupted participants. Else, if this is the case with $Q_{k,\varepsilon}$, then π_l is not revealed to the uncorrupted participants. Then, participants verify which random value enables to consecutively elect corrupted leaders until the round $k+t-1$. If this is the case with Q_k, then π_l is revealed to the uncorrupted participants. Else, if this is the case with $Q_{k,\varepsilon}$, then π_l is not revealed to the uncorrupted participants. And so on, until the round $k+1$. In the case where none participant in $\mathcal{A}$ is elected as leader at round $k+1$ neither with Q_k nor with $Q_{k,\varepsilon}$, then π_l is revealed to the uncorrupted participants.

Thus, the adversary $\mathcal{A}$ can win the t-forward unpredictability game of Section 6.2.2 with probability $\frac{f}{n}((\frac{f}{n})^t + (1-(\frac{f}{n})^t)(\frac{f}{n})^t) + (1-(\frac{f}{n})^t)\frac{1}{n-f} \cdot \frac{1}{n^t}$ instead of $(\frac{f}{n} + (1-\frac{f}{n})\frac{1}{n-f})\frac{1}{n^t}$.

6.5 Summary on Algorand

In this chapter, we analyzed the unpredictability properties in Algorand leader election protocol [57]. In particular, we showed that the unpredictability property defined in [15] is satisfied by Algorand. Then, we generalized this unpredictability property to the t-forward unpredictability to capture the case where an adversary elected as leader cannot predict the leaders of the t following elections. Finally, we described a strategy enabling an adversary who corrupts f participants to predict the next leaders. Indeed, when the number of potential leaders at each round is set to $n_l = 1$, this strategy increases the probability that the leader is corrupted and she predicts the t next leaders to $\frac{f}{n}((\frac{f}{n})^t + (1-(\frac{f}{n})^t)(\frac{f}{n})^t) + (1-(\frac{f}{n})^t)\frac{1}{n-f} \cdot \frac{1}{n^t}$ instead of $(\frac{f}{n} + (1-\frac{f}{n})\frac{1}{n-f})\frac{1}{n^t}$ with at most f participants are corrupted among the n participants. Therefore, the Algorand's parameter n_l originally sets in $\mathcal{Z}^+$ [28] has to be properly chosen. Indeed, n_l may be erroneously set to 1 for the sake of simplification and effectiveness of the consensus protocol as this enables to reduce the number of verification of potential leaders proofs, which is not a suitable choice. Note that, in the last implementation of the Algorand consensus protocol [2], n_l is set to 20.

Chapter 7

Security model and application to LEP-TSP

Contents

7.1 Overview

With the previous works provided in the last chapters, we define a security model of Single Leader Election (SLE) protocol with the five following security properties that aim to prevent well-known issues:

- The *uniqueness* property originally defined in [15] requires that exactly one leader is chosen in each election. This property is important to prevent forks where there are two or more valid leaders for an election. In a blockchain context, the fork may lead to two valid chains that extend the same block. As already seen in Chapter 4, this may delay the ledger consistency and impact its performance since participants converge to the same blockchain view only when the fork is resolved. Moreover, the fork may be leveraged to adopt rational strategies such as the selfish mining [44] as described in Chapter 4. The uniqueness property may be comparable to the common prefix property of Bitcoin backbone [49] that, however, does not prevent the fork in the last blocks.

- The *fairness* property defined in [15] means that each participant has a probability of $\frac{1}{n}$ of becoming the leader at each election. This ensures that each participant has the same chance to be leader at each election. Note that, for the sake of simplicity our fairness definition considers equal weighting of participants. However, it can easily be extended to consider participants with respect to some other distribution. For example, in a Proof-of-Stake consensus protocol, fairness could state that participants are selected as leader proportionally to the money they have invested. The fairness property may guarantee fair applications such as the random beacon described in Section 5.6. Unlike the fairness property, the chain quality property of Bitcoin backbone [49] does not prevent an adversary to increase her chance of being selected as leader during an advantageous chosen round. For this round, applications like a lottery which use the result of the leader election protocol as randomness source [18] may be unfair, *e.g.* increasing the chance of being the winner of the lottery.

- The *unpredictability* property defined in [15] means that an adversary who does not control the leader cannot learn which participants has been elected. This property aims to prevent an attacker from targeting the leader in DoS attacks, thus makes him unavailable to provide data to be added into the ledger. This property may also prevent a predictable leader from being bribed [17].

- The *t-forward unpredictability* property ensures that, even elected as leader, the adversary cannot predict the next leaders. This property generalizes the

unpredictability property [15] by capturing the case where the leader of the current election may be corrupted and the t following leaders have also to remain unpredictable. This property may prevent to bias the leader election process. For example, in some election protocol such as Algorand [28, 53, 29], a specific participant elected as leader for a given election may have an impact on following elections. This strategy is described in Chapter 6 Section 6.4. In a blockchain context, an adversary who knows at which election she will be elected leader can plan the data to add at a suitable time into the ledger. For example, this may be advantageous for high frequency trading if transactions are added at a suitable time into the ledger.

- The γ-*liveness* property as defined in [7] requires that, even if a fraction γ of participants is controlled by an adversary, it is still possible to elect a leader. This is a basic property of leader election protocols which guarantees new data to be continually added into the ledger. For example, this prevents an attacker from making a ledger useless due to the inability to add new data into a ledger. For example, in a cryptocurrency application, an attacker can impact the currency value by making the ledger useless with a DDoS for example and win easily money (as described in Chapter 4). Guaranteeing the γ-liveness property can prevent this type of attack. Thus, in a blockchain context, the liveness property defined in this chapter ensures that one block is added into the ledger at each election compared to the chain growth property of Bitcoin backbone [49] that does not guarantee this.

Then, we propose a SLE construction called LEP-TSP which is a new leader election protocol between n participants using one or several external Trusted RNG Service Providers (TSP) based on [54]. Every TSP uses the random number generator (RNG) of a Hardware Security Module (HSM). These TSPs act as external service providers which generate and provide random values, and they are not involved in the leader election process. This enables to reduce computation cost to elect a leader compared to the Bitcoin PoW consensus protocol [85] or the multiparty coin flipping protocol of Ouroboros [68]. Moreover, the TSPs assign exactly one random value at each participant for each election by tracking random values requests. Therefore, this strengthens the fairness of LEP-TSP to guarantee a uniform probability of being leader.

Our LEP-TSP protocol is designed for *permissioned* setting, *i.e.* the participants who take part in the protocol are authenticated. LEP-TSP operates while at most one third of participants are inactive or corrupted, *i.e.* entirely controlled and coordinated by an adversary. Our protocol meets the expected security properties in the random model oracle.

Broadly speaking, LEP-TSP proceeds as follows. It is divided in rounds. At each round k of LEP-TSP, one leader is elected among n participants. To this

end, each participant is associated to exactly one TSP provider. Note that, it is possible for several participants to share the same TSP as discussed in Section 7.3.1. First, each participant requests a random value from his TSP and broadcasts to other participants a participation value* computed with the obtained random value and a public value generated in the previous election. Then, the participant with the lowest participation value is selected as leader. Finally, the participants run several steps to acknowledge the same selected leader by collecting a threshold of acknowledgement messages. The leader elected at round k provides a contribution value B_k intended to be used in the election at round $k+1$.

7.2 Security model

We consider a group of n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ which use m external TSPs $\mathcal{TSP}_1, \ldots, \mathcal{TSP}_m$ to request random values. We also assume that the n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ agree to establish a certification authority $\mathcal{CA}$ that provides certificates to the participants and TSPs to authenticate each other.

In a Single Leader Election (SLE) protocol, the n participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ elect exactly one leader $\mathcal{P}_k$ at each round/election k. At every election k, each participant $\mathcal{P}_i$ requests a random value r_i from his TSP $\mathcal{TSP}_{a_i}$ and uses it to compute a participation value p_i. The participant with the lowest participation value is elected as leader.

7.2.1 SLE Model

Let λ be a security parameter. A SLE protocol is defined as a tuple of Probabilistic Polynomial Time (PPT) algorithms and protocols defined as follows: `SLE = (Setup, KeyGen, Register, RegisterVerify, Elect, Verify)`.

`Setup` protocol is a process intended to be run a single time before initiating a series of elections:

- `Setup(`1^λ`,param)`$\rightarrow$ `pp,st`$_0$. It takes as input the security parameter 1^λ and the set of parameters `param`, *e.g.* time intervals to collect messages or the threshold of acknowledgement for a leader. It generates the pair of secret and public keys $(sk_{\mathcal{CA}}, pk_{\mathcal{CA}})$ of the certification authority $\mathcal{CA}$. Then, it generates the pair of secret and public keys $(sk_{\mathcal{TSP}_a}, pk_{\mathcal{TSP}_a})$ of the TSP $\mathcal{TSP}_a$ and gets the certification $cert_{\mathcal{TSP}_a}$ for the public key $pk_{\mathcal{TSP}_a}$ from $\mathcal{CA}$. The initial

*As described in Section 7.3.4, the participants send in same time as their participation values, a value Q intended to be used in the next election. For example, in a blockchain context, Q may be a block of transactions.

participants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ agree on a first value B_0, *e.g.* B_0 may be the genesis block in a blockchain context. It initializes the list of registered participants $\mathcal{L}_0$. It outputs the public parameters $\mathtt{pp}$ and the initial state $\mathtt{st}_0$.

`KeyGen` algorithm is run by each participant before executing the registration process:

- `KeyGen` $(1^\lambda, \mathtt{pp}) \rightarrow sk, pk$. On input 1^λ and $\mathtt{pp}$, it generates a secret key sk and a public key pk.

`Register` protocol is run by each participant when he wants to start taking part in elections:

- `Register`$(i, \mathtt{pp}, k, \mathtt{st}_k, sk_i, pk_i) \rightarrow \mathtt{st}_{k+1}$. Each participant registers to take part in elections using a unique public identity i, the public parameter $\mathtt{pp}$, the index of the round k, the corresponding state $\mathtt{st}_k$, the secret key sk_i and the public key pk_i. It selects one authorized TSP $\mathcal{TSP}_{a_i}$ and gets a certification $cert_i$ for the tuple $(i, pk_i, pk_{\mathcal{TSP}_{a_i}})$ from the certification authority $\mathcal{CA}$. It modifies the state $\mathtt{st}_k$ to $\mathtt{st}_{k+1}$ by adding the tuple $(i, pk_i, pk_{\mathcal{TSP}_{a_i}}, cert_i)$ to the state $\mathtt{st}_k$.

Every time a new participant is registered, each previously registered participant has to run `RegisterVerify` algorithm in order to verify that the registration was correctly carried out:

- `RegisterVerify`$(i, \mathtt{pp}, k, \mathtt{st}_k, \mathtt{st}_{k-1}) \rightarrow 0/1$. Verification of a new participant $\mathcal{P}_j$'s registration from the perspective of the participant $\mathcal{P}_i$ is done using the public parameters $\mathtt{pp}$, the index of the round k, the corresponding state $\mathtt{st}_k$ and the previous state $\mathtt{st}_{k-1}$. It outputs 1 if the state $\mathtt{st}_k$ is valid and 0 otherwise.

At each round, `Elect` protocol is run once by each participant to elect a leader. During `Elect` process, several steps occur and the participants interact to agree on a leader as follows:

- `Elect`$(i, \mathtt{pp}, k, \mathtt{st}_k, sk_i, B_{k-1}) \rightarrow (0, \perp)/(1, \pi)$. It takes as inputs the public parameters $\mathtt{pp}$, the index of the round k, the corresponding state $\mathtt{st}_k$, the secret sk_i and the public value B_{k-1} generated in the previous election. Each participant $\mathcal{P}_i$ requests a random value r_i from his TSP $\mathcal{TSP}_{a_i}$ and computes the participation value p_i with the values r_i and B_{k-1}. Then, the participant $\mathcal{P}_i$ broadcasts p_i. The participant with the lowest participation value is selected as leader. Finally, several steps occur to collect a threshold of acknowledgement for the same leader. If the threshold is reached, then it outputs $(1, \pi)$ where π is the leader eligibility proof generated with the acknowledgement messages. Else, it outputs $(0, \perp)$.

`Verify` algorithm is used to check whether the participant selected as leader has been correctly elected.

- `Verify`(i,`pp`,k,$\texttt{st}_k$,B_{k-1},π)$\rightarrow 1/0$. Given an index i, the public parameters `pp`, the index of the round k, the corresponding state $\texttt{st}_k$, the contribution value B_{k-1} and a proof π claiming that a particular participant $\mathcal{P}_i$ was elected leader, the verification algorithm accepts or rejects the proof π that $\mathcal{P}_i$ has been elected leader.

7.2.2 Security properties

Before describing the security properties of a SLE protocol, we provide a generic experiment that applies for the five defined security properties.

Generic experiment

The generic experiment is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ as follows.

- **Setup Phase:** The adversary $\mathcal{A}$ selects the set of parameters `param`. The challenger $\mathcal{C}$ runs `Setup`(1^λ, `param`) and gives ($\texttt{pp}$,$\texttt{st}_0$) to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ selects the number of corrupted participants f and a related subset M of f indexes. The challenger $\mathcal{C}$ executes `KeyGen`(1^λ,`pp`) for the uncorrupted participants and gets the corresponding values (sk_i, pk_i).

- **Election Phase:** To register an uncorrupted participant, the adversary $\mathcal{A}$ sends the index i of the participant to the challenger $\mathcal{C}$. Then, $\mathcal{C}$ runs `Register`(i,`pp`,k,$\texttt{st}_k$,sk_i, pk_i) and it gives back to the adversary $\mathcal{A}$ the state $\texttt{st}_{k+1}$. To register a corrupted participant, the adversary $\mathcal{A}$ sends the index i of the participant and an updated state $\texttt{st}_{k+1}$ to the challenger $\mathcal{C}$. In either case, the challenger $\mathcal{C}$ runs `RegisterVerify`(j,`pp`,$k+1$,$\texttt{st}_{k+1}$,$\texttt{st}_k$) for any previously registered participant $\mathcal{P}_j$ which is an uncorrupted participant. If it ever happens that there is a call to `RegisterVerify`(j,`pp`,$k+1$,$\texttt{st}_{k+1}$,$\texttt{st}_k$) that returns 0, then the game immediately ends with output 0. Otherwise, the state is updated to $\texttt{st}_{k+1}$. The number of registered participants is denoted by n.
  During this phase, several elections may occur. For each new election, $\mathcal{C}$ runs on behalf of each uncorrupted participant $\mathcal{P}_i$ the algorithm `Elect`(i,`pp`,k,$\texttt{st}_k$, sk_i, B_{k-1}) and $\mathcal{C}$ sends the output $(1, \pi)$ or $(0, \perp)$ to $\mathcal{A}$.

After the election phase, a challenge phase is executed and it is specifically defined for each property.

Uniqueness

The *uniqueness* property requires that exactly one leader is chosen at each election. It means that exactly one participant in an election can prove that he is the elected leader. The uniqueness game originally defined in [15] does not consider the case where at a given election k, an adversary can corrupt specific participants in order to create a fork from previous elections $k' \leq k$. Indeed, in some elections such as Algorand, the adversary can start a fork from a previous round in order to create a longer chain than the main chain [100]. Thus, due to the longest rule, the adversary can rewrite the history. Therefore, we adapt the experiment of the uniqueness property defined in [15] to fit with the LEP-TSP protocol and to capture this specific case.

We denote by $\texttt{UNIQUE}[\mathcal{A}, \lambda, f, n]$ the uniqueness experiment which is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** The adversary $\mathcal{A}$ outputs values (b_j, π_j) for each corrupted participant. The experiment outputs 0 if for the election at the round k with the state $\texttt{st}_k$, each participant $\mathcal{P}_{i'}$ outputs the same value $(1, \pi_i) \leftarrow \texttt{Elect}(i', \texttt{pp}, k', \texttt{st}_{k'}, sk_{i'}, B_{k'-1})$ such that $\texttt{Verify}(i, \texttt{pp}, k', \texttt{st}_{k'}, \pi_i, B_{k'-1}) = 1$ for each $k' \leq k$. Otherwise, the experiment outputs 1.

We say that a SLE protocol is *unique* if no PPT adversary $\mathcal{A}$ can win the uniqueness game except with a negligible probability. That is, for all PPT $\mathcal{A}$, we have:

$$Pr\Big[\texttt{UNIQUE}[\mathcal{A}, \lambda, f, n] = 1\Big] \leq \texttt{negl}(\lambda)$$

Fairness

The property of *fairness* means that each participant has a probability of $\frac{1}{n}$ of being elected as the leader. We recall the fairness game already revisited in Chapter 5 that considers that even in the set of corrupted participants, each participant has a probability $\frac{1}{n}$ of being elected as the leader.

We denote the fairness experiment by $\texttt{FAIR}[\mathcal{A}, \lambda, f, n]$ which is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** At some point, all participants are registered. Then, the adversary $\mathcal{A}$ selects an participant $\mathcal{P}_{i'}$ and one more election occurs. Let $\mathcal{P}_i$ be the winner of this election. The experiment outputs 1 if and only if $i = i'$. Else, it outputs 0.

We say that a SLE protocol is *fair* if no PPT adversary $\mathcal{A}$ can win the fairness game with greater than negligible advantage. That is, for all PPT adversary $\mathcal{A}$, we have:

$$\left| Pr\Big[\texttt{FAIR}[\mathcal{A}, \lambda, f, n] = 1\Big] - \frac{1}{n} \right| \leq \texttt{negl}(\lambda)$$

Unpredictability

The property of *unpredictability* [15] means that an adversary $\mathcal{A}$ who does not control the leader cannot learn which participant has been elected. We adapt the refined unpredictability game of Chapter 5 to fit better with an LEP-TSP protocol.

We denote by $\texttt{UNPRED}[\mathcal{A}, \lambda, f, n]$ the unpredictability experiment. The experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge Phase.** At some point, all participants are registered and one more election occurs. In this election, the challenger $\mathcal{C}$ does not send the participation value p_j for each uncorrupted participant $\mathcal{P}_j$ to the adversary $\mathcal{A}$. Let $\mathcal{P}_i$ be the winner of this election. The game ends with $\mathcal{A}$ outputting an index i'. If $\mathcal{P}_i$ is a corrupted participant or if $\mathcal{P}_i$ is the single uncorrupted participant registered in the list $\mathcal{L}$, then the experiment outputs 0. Otherwise, it outputs 1 if and only if $i = i'$. By default, the output is 0.

We say that a SLE protocol is *unpredictable* if no PPT adversary $\mathcal{A}$ can win the unpredictability game with greater than negligible advantage when the winner of the election is uncorrupted. That is, for all PPT adversary $\mathcal{A}$, for any $f \leq n-2$, we have:

$$Pr\Big[\texttt{UNPRED}[\mathcal{A}, \lambda, f, n] = 1 \quad | \quad i \in \{1, \ldots, N\} \setminus M\Big] \leq \frac{1}{n-f} + \texttt{negl}(\lambda)$$

t-forward unpredictability

The *t-forward unpredictability* ensures that even elected as leader, the adversary cannot predict the next leaders. We recall the t-forward unpredictability game introduced in Chapter 6.

The t-forward unpredictability experiment is denoted by $\texttt{FUNPRED}[\mathcal{A}, \lambda, n, f]$ and is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge phase.** At some point, all participants are registered. Then, an election occurs. In this election, the challenger $\mathcal{C}$ does not send the participation value p_j for each uncorrupted participant $\mathcal{P}_j$ to $\mathcal{A}$. Let $\mathcal{P}_{i_0}$ be the winner of this election and $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_t}$ be the winners of the t following elections. The game ends with the adversary $\mathcal{A}$ outputting indexes $i'_0, i'_1, \ldots, i'_t$. It outputs 1 if and only if $i_0 = i'_0, i_1 = i'_1, \ldots, i_t = i'_t$. Else, it outputs 0.

We say that a SLE protocol is *t-forward unpredictable* if no PPT adversary $\mathcal{A}$ can win the t-forward unpredictability game with greater than negligible advantage. For all PPT adversary $\mathcal{A}$, we have:

$$Pr[\texttt{FUNPRED}[\mathcal{A}, \lambda, n, c] = 1] \leq (\frac{f}{n} + (1 - \frac{f}{n})\frac{1}{n-f})\frac{1}{n^t} + negl(\lambda)$$

Liveness

The γ-liveness property as defined in [7] requires that even if a fraction γ of participants is controlled by an adversary $\mathcal{A}$, it is still possible to elect a leader. We adapt the liveness game already defined in Chapter 5 to fit better with an LEP-TSP protocol.

We denote by α the fraction of corrupted participants and β the fraction of network fault, *i.e.* the fraction of uncorrupted participants whose communication is controlled by the adversary $\mathcal{A}$. The liveness experiment $\texttt{LIVE}[\mathcal{A}, \lambda, n, \alpha, \beta]$ is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows.

- **Challenge phase.** The adversary $\mathcal{A}$ outputs values (b_j, π_j) for each corrupted participant. The experiment outputs 0 if, for the election at the round k with the state $\texttt{st}_k$, at least one participant $\mathcal{P}_{i'}$ outputs $(1, \pi_i) \leftarrow \texttt{Elect}(i', \texttt{pp}, k, \texttt{st}_k, sk_{i'}, B_{k-1})$ such that any participant who verifies the pair $(1, \pi_i)$ outputs $1 \leftarrow \texttt{Verify}(i, \texttt{pp}, k, \texttt{st}_k, \pi_i, B_{k-1})$. Otherwise, the experiment outputs 1.

We say that a SLE protocol is δ-*live*, with $\alpha + \beta < \delta$, if no PPT adversary $\mathcal{A}$ can win the liveness game except with a negligible probability. That is, for all PPT adversary $\mathcal{A}$, we have:

$$Pr[\texttt{LIVE}[\mathcal{A}, \lambda, n, \alpha, \beta] = 1] \leq negl(\lambda)$$

7.3 LEP-TSP leader election construction

In this section, we first briefly introduce the prerequisites and assumptions for our protocol. Then, we detail our construction of LEP-TSP protocol.

7.3.1 Prerequisites and assumptions

In this section, we detail the assumptions about n and m which are the numbers of registered participants and TSPs respectively.

Trusted RNG service providers (TSP)

In our LEP-TSP protocol, n participants use m external TSP to request random values. Each TSP is an external service that provides exactly one random value per election and per participant. A tracking list may be stored in each TSP to register who requests a random for which election. Thus, this may prevent to deliver more than once time a random value to a participant for a given election. Each TSP has to be certified by the certification authority to be used in the protocol. We assume that TSPs generate trusted random values using a HSM and cannot be

compromised. Each TSP uses a trusted timer functionality which returns the current round number.

The number of TSPs in LEP-TSP is set to $2 \leq m \leq n$. Indeed, $m > 1$ enables to prevent the centralization to one component that may lead to a single point of failure issue. The upper bound $m \leq n$ may limit the number of TSP by participant to one. Indeed, since each TSP is limited to deliver one random value per election and per participant, a participant can request several random values from different TSPs and then chooses the one enabling to increase her chance to win the election.

It is possible that some TSP are not used. In the case where a TSP provider becomes unavailable or is corrupted during a period of time, it is necessary to exclude it from the list of authorized TSP providers used in the protocol. The participants using this useless or corrupted TSP have to be registered with another authorized one. The detection and replacement of useless TSP provider is left for future work.

Two third of honest participants

We assume that at least $\frac{2n}{3} + 1$ participants in LEP-TSP are honest, *i.e.* performing all the protocol instructions. Two third of honesty is the common assumption of the Byzantine fault tolerant (BFT) protocols [75, 24] enabling to guarantee security properties such as safety and liveness properties. Our protocol inherits these two third of honest participants assumption which also enables to guarantee the desired security properties and in particular the uniqueness and liveness properties under different network assumptions. The liveness is achieved when more than $\frac{2n}{3} + 1$ participants are perfectly capable of sending and receiving messages and the uniqueness is guaranteed even with a partitioned network between the $\frac{2n}{3} + 1$ honest participants. Note that, there exists consensus protocols assuming just a half of honesty. However, to the best of our knowledge, these protocols are vulnerable to fork issues that LEP-TSP prevents.

7.3.2 Validators selection and acknowledgement threshold

At each election of our LEP-TSP protocol, the participant with the lowest participation value is selected as leader and has to be acknowledged by a threshold of participants. In our LEP-TSP protocol, the acknowledgement of a leader is based on Algorand committee selection [29]. It is done through multiple subsets of participants named *validators* instead of all participants in order to improve liveness requirement.

Broadly speaking, at each step $s \geq 2$ of an election k, a validators set $VS_{k,s}$ of expected size n_v is randomly and secretly selected among all participants, *i.e.* their identity remains hidden until they send their acknowledgement for the selected leader. At step 2, the set $VS_{k,2}$ selects the participant with the lowest participation

value as leader. At steps $s \geq 3$, the set $VS_{k,s}$ acknowledges the selected leader if the threshold $th = \frac{2n_v}{3} + 1$ of acknowledgement for the same leader is received from $VS_{k,s-1}$. We refer to [29, 53] for the choice of n_v enabling a reasonable trade-off between liveness and uniqueness properties[†].

A participant $\mathcal{P}_i$ runs the following steps to learn if he is selected in the validators set $VS_{k,s}$ at step s of the election k:

- Send to his TSP $\mathcal{TSP}_{a_i}$ a request message $req_{i,k,s} = (k, i, cert_i, \texttt{RNGack}, s, rsign_i)$ where $rsign_i = SIGN[sk_i](i, k, \texttt{RNGack}, s)$. If $req_{i,k,s}$ is valid and is the first request for this step s of the election k, then the TSP $\mathcal{TSP}_{a_i}$ saves $req_{i,k,s}$, generates a random value $r_{i,s}$ and returns to $\mathcal{P}_i$ the message $asg_{i,k,s} = (k, req_{i,k,s}, r_{i,s}, asign_i)$ where $asign_i = SIGN[sk_{\mathcal{TSP}_{a_i}}](k, req_{i,k,s}, r_{i,s})$.
- Compute the target $pv_k = \frac{n_v}{|\mathcal{L}_{k-lb}|}$ where lb is a look back parameter to choose the set of participants for this election k.
- If $HASH(B_{k-1}||r_{i,s}||s) \leq pv_k$, then $\mathcal{P}_i \in VS_{k,s}$, *i.e.* $\mathcal{P}_i$ is selected as validator for the step s of the round k.

At each step where the participant $\mathcal{P}_i$ is selected in the set of validators $VS_{k,s}$, he broadcasts his vote according to the specification of the protocol (as described in Section 7.3.4) along with his validator eligibility proof $\pi_{i,k,s} = (asg_{i,k,s})$.

7.3.3 Lowest value rule

At each election, each participant requests a random value r from his TSP provider and uses it to compute a participation value p. Then, the participant with the lowest participation value is elected as leader. Compared to Algorand with a specific setting and SSLE, the lowest value rule enables to guarantee that at least one participant is selected as leader at each election, assuming that there is at least one honest participant registered in the leader election protocol. Indeed, in Algorand and SSLE, if the leader is corrupted, it is possible that there is no leader for this election. We choose also this lowest value rule to prevent a waste of time as in the PoET-based protocol where each participant has to wait a waiting time before proposing a new block.

7.3.4 Protocol description

At each round k of our LEP-TSP protocol, one participant is elected as leader among the n participants $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$. Let $HASH$ be a hash function and $S = (KGEN, SIGN, VERIF)$ be a signature scheme.

[†]In particular, n_v may be suitable chosen to guarantee uniqueness condition except with a probability of 5×10^{-9} [29].

- `Setup(`1^λ`,param)`. Parse the set of parameters `param`= $\{t_2, max, \{t_s\}_{s=3}^{max}, th, m, n_v, lb\}$ where t_2 is the time interval to collect the participation values, max is the maximal number of steps in an election, $\{t_s\}_{s=3}^{max}$ are times to collect the acknowledgement messages at each step $3 \leq s \leq max$, th is the number of acknowledgement messages to agree on the selected leader, m is the number of TSPs used in the protocol, n_v the expected number of validators in each step of an election k and lb is the look back parameter to select the validators set.

  The certification authority $\mathcal{CA}$ is set up outside from the n participants and uses $KGEN$ to get the secret key $sk_{\mathcal{CA}}$ and the corresponding public key $pk_{\mathcal{CA}}$. Next, $KGEN$ is run to obtain the pairs of secret and public keys $(sk_{\mathcal{TSP}_1}, pk_{\mathcal{TSP}_1}), \ldots, (sk_{\mathcal{TSP}_m}, pk_{\mathcal{TSP}_m})$ for the m TSPs $\mathcal{TSP}_1, \ldots, \mathcal{TSP}_m$. For each public key $pk_{\mathcal{TSP}_a}$, the pair $(\mathcal{TSP}_a, pk_{\mathcal{TSP}_a})$ is sent to the certification authority $\mathcal{CA}$ which returns the certification $cert_{\mathcal{TSP}_a} = SIGN[sk_{\mathcal{CA}}](\mathcal{TSP}_a, pk_{\mathcal{TSP}_a})$. For each TSP $\mathcal{TSP}_a$, a tracking list $\mathcal{TL}_{\mathcal{TSP}_a} = \{\}$ is initialized.

  Then, a first value B_0 agreed by the n participants is generated. The list of registered participants is initialized $\mathcal{L}_0 = \{\}$. The initial state is set as follows: $\mathtt{st}_0 = (B_0, \mathcal{L}_0, (\mathcal{TSP}_1, pk_{\mathcal{TSP}_1}, cert_{pk_{\mathcal{TSP}_1}}), \ldots, (\mathcal{TSP}_m, pk_{\mathcal{TSP}_m}, cert_{pk_{\mathcal{TSP}_m}}))$. It outputs `pp`= $(t_1, \{t_s\}_{s=2}^{max}, th, n_v, lb, max)$ and $\mathtt{st}_0$.

- `KeyGen (`1^λ`,pp)`. Run $KGEN$ to generate a secret key sk and the corresponding public key pk of the signature scheme $SIGN$.

- `Register(`i`,pp,`k`,`$\mathtt{st}_k$`,`sk_i, pk_i`)`. Parse the state $\mathtt{st}_k$ and the corresponding list $\mathcal{L}_k = ((1, pk_1, pk_{\mathcal{TSP}_{a_1}}, cert_1), (2, pk_2, pk_{\mathcal{TSP}_{a_2}}, cert_2), \ldots, (i-1, pk_{i-1}, pk_{\mathcal{TSP}_{a_{i-1}}}, cert_{i-1}))$. The participant $\mathcal{P}_i$ selects a public key $pk_{\mathcal{TSP}_{a_i}} \in \mathcal{L}_k$ of an authorized TSP $\mathcal{TSP}_{a_i} \in \mathtt{st}_k$. Then, $\mathcal{P}_i$ provides $(i, pk_i, pk_{\mathcal{TSP}_{a_i}})$ to the certification authority $\mathcal{CA}$ which returns $cert_i = SIGN[sk_{\mathcal{CA}}](i, pk_i, pk_{\mathcal{TSP}_{a_i}})$. The participant $\mathcal{P}_i$ updates $\mathtt{st}_k$ to $\mathtt{st}_{k+1}$ by adding $(i, pk_i, pk_{\mathcal{TSP}_{a_i}}, cert_i)$ to $\mathtt{st}_k$. It outputs $\mathtt{st}_{k+1}$.

- `RegisterVerify(`i`,pp,`k`,`$\mathtt{st}_k$`,`$\mathtt{st}_{k-1}$`)`. Parse the states $\mathtt{st}_{k-1}$ and $\mathtt{st}_k$. Verify $|\mathcal{L}_{k-1}| < |\mathcal{L}_k| + \frac{|\mathcal{L}_k|}{10}$. Parse $\mathtt{st}_k$ and recover the value $(j, pk_j, pk_{\mathcal{TSP}_{a_j}}, cert_j)$ of the new participant $\mathcal{P}_j$. The participant $\mathcal{P}_i$ verifies if there is no duplicate of both identity j and public key pk_j in $\mathtt{st}_k$. Then, $\mathcal{P}_i$ checks if $pk_{\mathcal{TSP}_{a_j}} \in \mathtt{st}_k$, $VERIF(pk_{\mathcal{CA}}, (\mathcal{TSP}_{a_j}, pk_{\mathcal{TSP}_{a_j}}), cert_{\mathcal{TSP}_{a_j}}) = 1$ and $VERIF(pk_{\mathcal{CA}}, (j, pk_j, pk_{\mathcal{TSP}_{a_j}}), cert_j) = 1$. If the above checks pass, then it outputs 1. Else, it outputs 0.

- `Elect(`i`,pp,`k`,st`$_k$`,`sk_i`,`B_{k-1}`)`. The election protocol is based on the Algorand leader election described in Chapter 6. The participant $\mathcal{P}_i$ runs the following steps to elect the leader of the round k.

  Step $s = 1$

  - $\mathcal{P}_i$ sends to his TSP $\mathcal{TSP}_{a_i}$ a request message $req_{i,k} = (k, i, cert_i,$ `RNGprt`$, rsign_i)$ where $rsign_i = SIGN[sk_i](i, k,$`RNGprt`$)$ and `RNGprt` is the tag to inform his TSP that $\mathcal{P}_i$ requests a random for computing his participation value.
  - $\mathcal{TSP}_{a_i}$ checks the following verifications: k is the index of the current round, $req_{i,k} \notin \mathcal{P}_{\mathcal{TSP}_{a_i}}$, $VERIF(pk_{\mathcal{CA}}, (i, pk_i, pk_{\mathcal{TSP}_{a_i}}), cert_i) = 1$ and $VERIF(pk_i, (i, k,$`RNGprt`$), rsign_i) = 1$. If the above checks pass, $\mathcal{TSP}_{a_i}$ adds $req_{i,k}$ to $\mathcal{TL}_{\mathcal{TSP}_{a_i}}$, generates a random value r_i and returns to $\mathcal{P}_i$ the assignment message $asg_{i,k} = (k, req_{i,k}, r_i, asign_i)$ where $asign_i = SIGN[sk_{\mathcal{TSP}_{a_i}}](k, req_{i,k}, r_i)$.
  - When the participant $\mathcal{P}_i$ receives $asg_{i,k}$ from his TSP provider, $\mathcal{P}_i$ verifies if $VERIF(pk_{\mathcal{TSP}_{a_i}}, (k, req_{i,k}, r_i), asign_i) = 1$. If yes, $\mathcal{P}_i$ computes his participation value $p_i = HASH(B_{k-1}||r_i)$ and chooses a value B_i. Then, $\mathcal{P}_i$ broadcasts to other participants the participation message $prt_i = (k, p_i, asg_{i,k}, B_i, psign_i)$ where $psign_i = SIGN[sk_i](k, p_i, asg_{i,k}, B_i)$.

  Step $s = 2$

  - During the interval t_2, the participant $\mathcal{P}_i$ collects the participation messages prt_j from other participants. If $\mathcal{P}_i \in VS_{k,2}$, then $\mathcal{P}_i$ sets the proof $\pi_{i,k,2}$ (as in Section 7.3.2) and verifies if there is exactly one valid participation message received from a participant $\mathcal{P}_j$. Otherwise, $\mathcal{P}_i$ discards the participant $\mathcal{P}_j$ from the election. Next, $\mathcal{P}_i$ selects one participation message $prt_l = (k, p_l, asg_{l,k}, B_l, psign_l)$ such that
    - $asg_{l,k} = (k, req_{l,r}, r_l, asign_l)$ is valid: the index k is the current round, $req_{l,r}$ is valid, $VERIF(pk_{\mathcal{CA}}, (k, req_{l,k}, r_l), asign_l) = 1$ and $(l, pk_l, pk_{\mathcal{TSP}_{a_l}}, cert_l) \in$ `st`$_k$
    - the participation value is computed with the value B_{k-1} resulting from the previous election and the random value provided by the corresponding TSP $\mathcal{TSP}_{a_l}$: $p_l = HASH(B_{k-1}||r_l)$
    - the signature is valid: $VERIF(pk_l, (k, p_l, asg_{l,k}, B_l), psign_l) = 1$
    - the participation value is the lowest one received: $p_l \leq p_j$ for all p_j included in the received participation messages prt_j.

    The participant $\mathcal{P}_i$ sets $vote = prt_l$ and broadcasts his acknowledgement message $ack_{i,k,2} = (k, vote, asg_{i,k,2}, \pi_{i,k,2}, acksign_i)$ where $acksign_i =$

$SIGN[sk_i](k, 2, vote, \pi_{i,k,2})$. Else, if these above verification do not pass or if $\mathcal{P}_i$ receives none participation value, then $\mathcal{P}_i$ broadcasts an acknowledgement message for $vote = \varepsilon$.
Then, in either case[‡], $\mathcal{P}_i$ starts Step 3.

Step $s = 3$

– the participant $\mathcal{P}_i$ collects the acknowledgement messages ack_j from the participants in $VS_{k,2}$ during the time t_3. If $\mathcal{P}_i \in VS_{k,3}$, then $\mathcal{P}_i$ sets the proof $\pi_{i,k,3}$ and executes the following actions. If $\mathcal{P}_i$ receives at least $\frac{2n_v}{3} + 1$ valid acknowledgement messages ack_j from different validators in $VS_{k,2}$ for the same value $vote \neq \varepsilon$, then $\mathcal{P}_i$ broadcasts $ack_{i,k,3} = (k, vote, asg_{i,k,3}, \pi_{i,k,3} acksign_i)$ where $acksign_i = SIGN[sk_i](k, 3, vote, \pi_{i,k,3})$. Else, $\mathcal{P}_i$ broadcasts an acknowledgement message for the value $vote = \varepsilon$.
Then, in either case, $\mathcal{P}_i$ starts Step 4.

Step $s = 4$

– the participant $\mathcal{P}_i$ collects the acknowledgement messages ack_j from the participants in $VS_{k,3}$ during the time t_4. If $\mathcal{P}_i \in VS_{k,4}$, then $\mathcal{P}_i$ sets the proof $\pi_{i,k,4}$ and executes the following actions. If $\mathcal{P}_i$ receives at least $\frac{2n_v}{3} + 1$ valid acknowledgement messages ack_j from different validators in $VS_{k,3}$ for the same value $vote \neq \varepsilon$, then $\mathcal{P}_i$ broadcasts a certify message $cm_{i,k,4} = (0, vote, asg_{i,k,4}, \pi_{i,k,4}, certsign_i,)$ where $certsign_i = SIGN[sk_i](k, 4, vote, \pi_{i,k,4})$. Else, the participant $\mathcal{P}_i$ broadcasts a certify message $cm_{i,k,4} = (1, vote, asg_{i,k,4}, \pi_{i,k,4}, certsign_i)$ for the value $vote$ defined as follows:

  * $vote = \varepsilon$ if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3} + 1$ of valid acknowledgement messages $ack_{j,k,3} = (k, vote, asg_{j,k,3}, acksign_j, \pi_{i,k,3})$ for the same value $vote = \varepsilon$ from different participants in $VS_{k,3}$.
  * any $vote \neq \varepsilon$ if $\mathcal{P}_i$ receives at least $\frac{n_v}{3} + 1$ of valid acknowledgement messages $ack_{j,k,3} = (k, vote, asg_{j,k,3}, acksign_j, \pi_{i,k,3})$ for the same value $vote \neq \varepsilon$ from different participants in $VS_{k,3}$.
  * $vote = \varepsilon$ otherwise

  Then, in either case, $\mathcal{P}_i$ starts Step 5.

Step $5 \leq s \leq m - 1$

[‡]This means that even if the participant is selected or not in the validators set of this step, he starts the following step by waiting the messages from the validators of this step.

– the participant $\mathcal{P}_i$ collects the acknowledgement messages ack_j from the participants $VS_{k,s-1}$ during the time t_s. If $\mathcal{P}_i \in VS_{k,s}$, then $\mathcal{P}_i$ sets the proof $\pi_{i,k,s}$ and executes the following actions.

  * *Ending Condition 0:* at any step $5 \leq s' \leq s$, if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3}+1$ valid certify messages $cm_{j,k,s'-1} = (0, vote, asg_{j,k,s'-1}, \pi_{i,k,s'-1}, certsign_j)$ for the same value $vote = prt_l$, then he sets $\pi = (k, l, prt_l, ACK_l)$ where $ACK_l = \{th$ certify messages for $prt_l\}$, and $B_k = B_l$. It outputs $(1, \pi)$.
  * *Ending Condition 1:* else, at any step $5 \leq s' \leq s$, if $\mathcal{P}_i$ receives at least $\frac{2n_v}{3}+1$ valid certify messages $cm_j = (1, vote, asg_{j,k,s'-1}, \pi_{i,k,s'-1}, certsign_j)$ for any value $vote_j$, then set $B_k = B_\varepsilon$ where B_ε is the empty value. Set $\pi_\varepsilon = (k, \varepsilon, ACK_\varepsilon)$ where $ACK_l = \{th$ certify messages for any value$\}$. It outputs $(1, \pi_\varepsilon)$.
  * else, $\mathcal{P}_i$ recovers the value $vote$ as in the step 4. The participant $\mathcal{P}_i$ broadcasts $cm_i = (1, vote, asg_{i,k,s}, certsign_i, \pi_{i,k,s})$ if he has received more than $\frac{2n_v}{3}+1$ valid certify messages of the form $cm_j = (1, vote_j, asg_{j,k,s-1}, \pi_{i,k,s-1}, certsign_j)$ for any value $vote_j$. Else, $\mathcal{P}_i$ broadcasts $cm_i = (0, vote, asg_{i,k,s}, \pi_{i,k,s}, certsign_i)$. Then, $\mathcal{P}_i$ starts Step $s+1$.

Step $s = m$ [§]

– If $\mathcal{P}_i \in VS_{k,m}$, then $\mathcal{P}_i$ sets the proof $\pi_{i,k,m}$ and executes the following actions. The participant $\mathcal{P}_i$ verifies *Ending Condition 0.* Else, $\mathcal{P}_i$ verifies *Ending Condition 1.* Else, he sets $vote = \varepsilon$ and broadcasts $cm_i = (1, vote, asg_{i,k,s}, \pi_{i,k,m}, certsign_i)$.

If $\mathcal{P}_i \notin VS_{k,s}$ or the step m does not enable to output: at any step $s \geq 5$, $\mathcal{P}_i$ enters in *Ending condition 1* or *Ending condition 0.* Else, it outputs $(0, \perp)$.

- `Verify(`i`,pp,`k`,st`$_k$`,`π`,`B_{k-1}`)`. If $\pi \neq \pi_\varepsilon$, parse the proof $\pi = (k', i, prt_i, ACK_i)$ and $prt_i = (k', p_i, asg_{i,k'}, B_i, psign_i)$. Verify if $k - 7 \leq k' \leq k$. Then, check if $(i, pk_i, pk_{\mathcal{TSP}_{a_i}}, cert_i) \in \mathtt{st}_{k'}$ and $VERIF(pk_{\mathcal{CA}}, (i, pk_i, pk_{\mathcal{TSP}_{a_i}}), cert_i) = 1$. Verify if prt_i is a valid participation message and ACK_i contains at least th valid acknowledgement messages for the same participation message prt_i. If the above verifications pass, set $B_k = B_i$ and output 1. If $\pi = \pi_\varepsilon$, verify if ACK_ε contains at least th valid acknowledgement messages for certify messages for any value. If yes, output 1. Else output 0.

[§] m is chosen such that with overwhelming probability, `Elect` protocol has ended before this step m [28, 29]

7.4 Security analysis

We call the *winner random value* the random value r provided by the TSP provider enabling to obtain the lowest participation value p. Therefore, the participant with the winner random value is elected as leader.

Hereinafter, we analyze the uniqueness, fairness, unpredictability, t-forward unpredictability and liveness of the LEP-TSP protocol. Let $HASH$ be a random oracle.

7.4.1 Uniqueness: security proof

In this section, we prove the following theorem:

Theorem 5 *For any adversary $\mathcal{A}$ which corrupts $f \leq \frac{n}{3} - 1$ participants, the LEP-TSP protocol provided in Section 7.3.4 is unique under the unforgeability of the signature scheme and the unpredictability and randomness of the random number generator in the random oracle model.*

Proof. The game is the following. At the election at the round k with the corresponding state $\mathtt{st}_k$, the adversary $\mathcal{A}$ runs $\mathtt{Elect}(j,\mathtt{pp},k,\mathtt{st}_k,sk_j,B_{k-1})$ for each corrupted participant $\mathcal{P}_j$ and get the value (b_j, π_j). The adversary $\mathcal{A}$ wins the uniqueness game (Section 7.2.2) when there is a fork at a round $k' \leq k$, *i.e.* at least two different participants output $(1, \pi_{i_1}) \leftarrow \mathtt{Elect}(j,\mathtt{pp},k,\mathtt{st}_k,sk_j,B_{k-1})$ and $(1, \pi_{i_2}) \leftarrow \mathtt{Elect}(j,\mathtt{pp},k,\mathtt{st}_k,sk_j,B_{k-1})$, respectively, such that $\mathtt{Verify}(i_1,\mathtt{pp},k', \mathtt{st}'_k,\pi_{i_1},B_{k'-1}) = \mathtt{Verify}(i_2,\mathtt{pp},k',\ \mathtt{st}'_k,\pi_{i_2},B_{k'-1}) = 1$ and so $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$ are two valid leaders of the round k'. Note that, two participants receive the same λ-bit value with a probability of $\frac{1}{2^\lambda}$ due to the birthday paradox since the values are generated across the same distribution. Several cases are possible to generate these proofs.

- *Case 1:* We assume that $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$ request their random values to the same TSP provider $\mathcal{TSP}_a$. The adversary $\mathcal{A}$ bias the RNG of $\mathcal{TSP}_a$ which assigns the same random value r to $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$ who obtain the same participation value p. Note that, the value r is the winner random value if the other random values provided by the TSP enable to obtain a participation value higher than p. This means that the adversary can distinguish with a non-negligible advantage the different values provided by the TSPs from a truly random value. The proofs π_{i_1} and π_{i_2} are considered valid if they receive the threshold th of valid certify messages from the validators set of a step $s \geq 4$. We denote by f_s the number of corrupted participants selected as validators of the step s. At a step $s \geq 4$, the adversary generates $th - c_s$

certify messages on behalf of $th - f_s$ uncorrupted participants for $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$. Note that, a certify message $cm_i = (0, vote, asg_{i,k',s}, \pi_{i,k',s}, certsign_i)$ is valid if $VERIF(pk_i, (0, vote, asg_{i,k',s}, \pi_{i,k',s}), certsign_i) = 1$. If the adversary $\mathcal{A}$ wins the game, it means that she has generated a valid random value for $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$, and $th - c_s$ valid certify messages that certify the leadership $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$. In this case, $\mathcal{A}$ has broken the randomness and unpredictability of the random number generator and the unforgeability of the signature scheme.

- *Case 2:* the adversary $\mathcal{A}$ generates a valid signature $asign_{i_2,k}$ for the corrupted participant $\mathcal{P}_{i_2}$ such that $VERIF(pk_{\mathcal{TSP}_{i_2}}, (k, req_{i_2,k'}, r_1), asign_{i_2}) = 1$ with r_1 the winner random value provided by the uncorrupted participant $\mathcal{P}_{i_1}$ during the interval t_2. As in Case 1, the adversary generates the $th - f_s$ certify messages on behalf of $th - f_s$ uncorrupted participants that certify the participant $\mathcal{P}_{i_2}$. In this case, this would imply that the adversary $\mathcal{A}$ has broken the unpredictability of the random number generator and the unforgeability of the signature scheme.

- *Case 3:* During the challenged election k, the adversary $\mathcal{A}$ creates a fork from the round $k' < k$ as in [100]. We denote by n_k the number of participants registered in the election at round k. We prevent the fork described in [100] with the following mechanisms:

  - the limitation of registration in `RegisterVerify` algorithm which accepts at most $n_k + \frac{n_k}{10}$ registered participants for the next round $k+1$. Thus, it is not possible from a round k to the next round $k+1$, n_k represents $\frac{n_{k+1}}{3} - 1$. In particular, we have $k' < k - 7$ such that $n_{k'}$ represents $\frac{n_k}{3} - 1$.
  - the proofs with an index $k' < k - 7$ are discarded in `Verify`. Therefore, the adversary can only create a fork from the round $k-7$. At each round k' from $k-7$ to k, the adversary generates π_{i_2} such that `Verify(`i_2`,pp,`k'`,` $\mathtt{st}'_k$`,`π_{i_2}`,`$B_{k'-1}$`)`$= 1$ and $i_2 \neq i_1$ where $\mathcal{P}_{i_1}$ is the leader of the round k'. If $\mathcal{A}$ wins the game, it means that $\mathcal{A}$ generates these proofs by either biasing the RNG of TSP provider to obtain a random value r_{i_2} enabling to obtain a lower participation value than p_{i_1} or forging a valid signature on a random value r_{i_2}. Moreover, it means that $\mathcal{A}$ generates valid certify messages on behalf $th - c_s$ uncorrupted participants selected as validators which certify the leadership of $\mathcal{P}_{i_2}$. This means that $\mathcal{A}$ has broken the unpredictability of the random number generator and the unforgeability of the signature scheme.
  - verification in `Elect` by TSP providers enable to provide a random value only for the current round. Thus, the adversary cannot create a longer

chain until the round $k+1$ during the challenged election k.

- *Case 4:* Since the LEP-TSP protocol uses acknowledgement and certify steps as in Algorand, the proof for the uniqueness property assuming that $\mathcal{A}$ corrupts $f \leq \frac{n}{3} - 1$ participants and the $n - f$ other participants may be partitioned is also applicable. Indeed, the idea is that even if the $n - f$ are partitioned in two groups, *w.l.o.g.*, then the probability that the threshold th is reached in each group to certify two different leaders is small. The details can be found in [28] at Section 10.

7.4.2 Fairness: security proof

In this section, we provide the proof of the following theorem:

Theorem 6 *For any adversary $\mathcal{A}$ which corrupts $f \leq n$ participants, the LEP-TSP protocol provided in Section 7.3.4 is fair under the unforgeability of the signature scheme, and the unpredictability and randomness of the random number generator in the random oracle model.*

Proof. Note that, the checks done in `RegisterVerify` ensure that each participant $\mathcal{P}_j$ with his unique identity j is registered once and only with one certified public key pk_j. A second registration under the same identity j and another public key pk'_j is rejected. Moreover, `RegisterVerify` ensures also that each certified key pk_j is associated to exactly one authorized TSP provider. Furthermore, all TSP providers keep track of participants' requests at each election. Each TSP provider registers the identity j, the corresponding public key pk_j and the certification $cert_j$ of each participant $\mathcal{P}_j$ who requests a random value for the election k. Thus, this ensures that only one random value at each election is assigned to each participant who would like to participate in the election.

Assume that the adversary $\mathcal{A}$ wins the fairness game defined in Section 7.2.2. This means that, at the election k, $\mathcal{A}$ has selected the participant $\mathcal{P}_{i'}$ such that $\texttt{Verify}(i', \texttt{pp}, k, \texttt{st}_k, \pi_{i'}, B_{k'-1}) = 1$. Several cases are possible to win the game.

- *Case 1:* the adversary $\mathcal{A}$ winning the fairness game means that it bias the RNG of the TSP provider $\mathcal{TSP}_{a_{i'}}$ which provides the winner random value $r_{i'}$ enabling to obtain the lowest participation value $p_{i'}$ for the participant $\mathcal{P}_{i'}$. This means that the adversary can distinguish with a non-negligible advantage the different values provided by other TSPs from a truly random value. In this case, this would mean that $\mathcal{A}$ has broken the randomness and unpredicatbility of the random number generator.

- *Case 2:* the adversary $\mathcal{A}$ can also generate a valid signature $asign'_{i',k}$ on a value r which enables to obtain the lowest participation value. Thus, $VERIF(pk_{\mathcal{TSP}_{a_{i'}}}, (k, req_{i',k,r}), asign'_{i',k}) = 1$. This means that the adversary can distinguish with a non-negligible advantage the different values provided by other TSPs from a truly random value. In this case, $\mathcal{A}$ has broken the unforgeability of the signature scheme and the unpredicatbility of the random number generator.

Therefore, each participant in the list $\mathcal{L}_k$ is registered only with one certified public key and is associated to exactly one TSP provider. Thus, at each election, each participant receives only one random value enabling to compute one participation value and has a probability of $\frac{1}{n}$ of being elected as leader.

7.4.3 Unpredictability: security proof

In this section, we prove the following theorem:

Theorem 7 *For any adversary $\mathcal{A}$ which corrupts $f \leq n-2$ participants, the LEP-TSP protocol described in Section 7.3.4 is unpredictable under the unpredictability of the random number generator in the random oracle model.*

Proof. Note that, a participant knows that he is the leader of the election at the end of the time t_2, *i.e.* when all participation values are revealed and his participation value is the lowest one.

Suppose that the adversary $\mathcal{A}$ wins the unpredictability game defined in Section 7.2.2. Then, at some point of the challenged election k, the adversary $\mathcal{A}$ outputs the index i' such that $\mathcal{P}_{i'}$ is an uncorrupted participant and `Verify(`i'`,pp,`k`,` $\mathtt{st}_k, \pi_{i'}, B_{k-1}$`)`$= 1$. It means that the adversary $\mathcal{A}$ can learn which value $r_{i'}$ provided by the TSP provider $\mathcal{TSP}_{i'}$ is the winner random value. Thus, the adversary $\mathcal{A}$ can guess with a non-negligible advantage the different outputs of TSPs and which one is the winner random value. This implies that the attacker has broken the unpredictability property of the random number generator.

Therefore, to predict the leader of the current election, the adversary cannot do better than randomly choosing an uncorrupted participant in the list $\mathcal{L}_k$ of registered participants, *i.e.* with a probability of at most $\frac{1}{n-f}$.

7.4.4 t-forward unpredictability: security proof

In this section, we provide the proof of the following theorem:

Theorem 8 *For any adversary $\mathcal{A}$ which corrupts $f \leq n$ participants, the LEP-TSP protocol provided in Section 7.3.4 is t-forward unpredictable under the unpredictability and randomness of the random number generator in the random oracle model.*

Proof Assume that the adversary $\mathcal{A}$ wins the t-forward unpredictability game defined in Section 7.2.2. This means that, at some point of the challenged election k, the adversary $\mathcal{A}$ outputs the indexes $i'_0, i'_1, \ldots, i'_t$ such that the participants $\mathcal{P}_{i'_0}, \mathcal{P}_{i'_1}, \ldots, \mathcal{P}_{i'_t}$ are the leaders of the challenged election k and the t following elections $k+1, \ldots k+t$, respectively. We can distinguish two possible cases: either $\mathcal{P}_{i'_0}$ is a corrupted participant, or $\mathcal{P}_{i'_0}$ is an uncorrupted participant.

- In the case where the leader $\mathcal{P}_{i'_0}$ is an uncorrupted participant, the adversary $\mathcal{A}$ predicting the leader $\mathcal{P}_{i'_0}$ means that it can guess with a non-negligible advantage which outputs are provided by TSP providers and which one is the winner random value. Note that, at each election, the participants have to request a new random value to compute their participation value to take part in the election. Thus, to predict the t next leaders, the adversary $\mathcal{A}$ guesses with a non-negligible advantage which outputs provided by TSP provider for the t next elections and which values $r_{i_1}, \ldots, r_{i_t}$ are the winner random values of the t next elections. In this case, the adversary $\mathcal{A}$ has broken the unpredictability property of the random number generator.

  To predict the t next leaders, the adversary $\mathcal{A}$ can also bias the RNG of a TSP provider which provides the winner random values $r_{i_1}, \ldots, r_{i_t}$ enabling to obtain the lowest participation values $p_{i'_0}, \ldots, p_{i'_t}$ for the participants $\mathcal{P}_{i'_1}, \ldots, \mathcal{P}_{i'_t}$. This means that the adversary $\mathcal{A}$ can guess with a non-negligible advantage the different outputs of TSPs and which one is the winner random value. In this case, $\mathcal{A}$ has broken the unpredictability and randomness properties of the random number generator.

- In the case where the leader is $\mathcal{P}_{i'_0}$ corrupted, the adversary $\mathcal{A}$ wins the game by predicting the t next leaders. Since new random values are requested at each election to compute participation values, thus the adversary $\mathcal{A}$ guesses with a non-negligible advantage which values $r_{i_1}, \ldots, r_{i_t}$ provided by the TSP providers are the winner random values for the t next elections. In this case, the adversary $\mathcal{A}$ has broken the unpredictability property of the random number generator.

  The adversary $\mathcal{A}$ can also bias the RNG of a TSP provider which provides the values $r_{i_1}, \ldots, r_{i_t}$ enabling to obtain small participation values $p_{i'_0}, \ldots, p_{i'_t}$ and guess with a non-negligible advantage the different outputs of TSPs for the t next elections. In this case, $\mathcal{A}$ has broken the unpredictability and randomness properties of the random number generator.

Thus, the adversary wins the unpredictability game with a probability of $(\frac{f}{n} + (1 - \frac{f}{n})\frac{1}{n-f})\frac{1}{n^t}$. Indeed, the leader of the current election is a corrupted participant with a probability of $\frac{f}{n}$. If the leader is not a corrupted participant,

then the adversary can do not better than choosing at random an participant among the uncorrupted participants registered in the list $\mathcal{L}_k$, *i.e.* she guesses the leader with a probability of at most $(1 - \frac{f}{n})\frac{1}{n-f}$. Given the current election, $\mathcal{A}$ predicts the t following leaders by randomly choosing t participants in the list $\mathcal{L}_k$ and thus predicts the t following leaders with probability of at most $\frac{1}{n^t}$.

7.4.5 Liveness: security proof

In this section, we prove the following theorem:

Theorem 9 *For any adversary $\mathcal{A}$ which corrupts $f \leq \frac{n}{3} - 1$, the LEP-TSP protocol provided in Section 7.3.4 is $\frac{1}{3}$-live in the random oracle model.*

Proof During an election k, any participant registered along with a valid certificate delivered by the certification authority $\mathcal{CA}$ is authorized to participate in the election. An authorized participant $\mathcal{P}_i$ requests a random value r_i to his TSP provider $\mathcal{TSP}_{a_i}$ and broadcasts his participation value $p_i = HASH(B_{k-1}||r_i)$ to participate in the election. Anyone before all participants reveal their participation value is aware of the winner, even the leader himself. In the case where the leader is a corrupted participant who chooses to not reveal her participation value, the participant with the second lowest value is the lowest one in the view of honest participants. Thus, this guarantees that at least one participation value p_i is broadcasted by an uncorrupted participant $\mathcal{P}_i$.

The adversary $\mathcal{A}$ wins the liveness game in the case where any participation value p_i reached the threshold th of acknowledgement messages from the sets of validators. The adversary $\mathcal{A}$ can corrupt $f = \alpha + \beta$ participants where α participants are entirely controlled and coordinated by $\mathcal{A}$ and β are uncorrupted participants whose $\mathcal{A}$ controls their communication, *i.e.* $\mathcal{A}$ can choose which messages are received or sent to other participants. As in Algorand [53], the liveness property is satisfied in LEP-TSP under the assumption that $f \leq \frac{n}{3} - 1$[¶]. Indeed, while the other $n - f \geq \frac{2n}{3} + 1$ participants are uncorrupted and perfectly capable of sending and receiving messages, a participation value p_i reaches the threshold th acknowledgement messages. The validators set size n_v of each step is sufficiently large [29, 53] to guarantee with high probability that the number of honest participants in $VS_{k,s}$ is strictly higher than the threshold th. The proof provided in [53] at Section C2 apply in our LEP-TSP.

[¶]In [53], this is equivalent to the *strong synchrony* definition that does not allow the adversary to manipulate the network at a large scale and does not allow network partitions.

7.5 Summary

In this chapter, we first defined a security model of a Single Leader Election protocol that addresses well-known issues and attacks targeting consensus protocols, seen all along of this first part on consensus protocols for blockchain. Then, we proposed a construction of LEP-TSP protocol [56] which is a new leader election protocol based on external RNG services, intended to be used in private setting. We proved that our LEP-TSP protocol meets the expected security properties of a SLE protocol. In particular, LEP-TSP operates while $f < \frac{n}{3}$ participants are corrupted by an adversary, with n the total number of participants. Some topics are left for the future work such as an improved model assuming compromised TSP providers or a performance analysis of our protocol.

Chapter 8

Useful work

Contents

8.1 Overview of useful work

We propose the Useful Work (UW) protocol which is a variant of the PoW mechanism where the work is used to solve *any* real world problem. Our UW

protocol is designed to be used in public setting, *i.e.* anyone at any time can participate in the protocol, and it is based on PoW and PoS mechanisms. We assume that while more than two thirds of coins are owned by honest participants, then our UW protocol works.

Broadly speaking, our UW protocol proceeds as follows. It is divided in rounds. At each round k, the participants compete to win *useful coins*. To this end, they have to perform the computational work of problems submitted by clients and provide a proof of their correct computations. When the computational works are validated by a group of verifiers, a random election is run to choose one of the participants to win useful coins. Then, a participant is elected to generate the block B_k of valid transactions that contains also the data related to the winner of useful coins.

This chapter is intended to give an insight on what we can do with computing power instead of using it to solve the Bitcoin PoW puzzle. This can be served as basis for further studies to construct a practical consensus protocol that solves any world problems.

8.2 Entities and building blocks

In this section, we present the different entities and the main building blocks of the UW protocol.

We assume that we have a random number generator that provides trusted and verifiable random values. Several works study how to generate such random values in distributed manner [23, 98, 94, 93] which is out of scope of this work.

8.2.1 Two type of entities

There are two types of entities in our Useful Work protocol: one or more clients and several participants.

Clients. The clients are external entities who need computing power for any real world problem, *e.g.* scientific experiments, mathematical problems, etc. The clients submit problems in the form of a code to run. Each client $\mathcal{C}_i$ owns an account $\mathtt{acc}_{\mathcal{C}_i}$ that contains an amount of useful coins related to a pair of public and secret keys $(pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i})$.

Participants. The participants perform the computational work of submitted problems and maintain the blockchain. They may have one or several roles: worker, voter, verifier and writer. For example, a worker may be also selected as a verifier or as a writer.

Workers perform computation works of submitted problems to win useful coins. To this end, each worker has to run the code of a submitted problem and provides a result along with a proof of correct computation. Voters are selected to define which are the most relevant problems to run. Verifiers are selected to verify the result published and/or the generated block. The verifiers may earn also useful coins for their verification. A writer is elected to generate a block of valid transactions and add it into the blockchain. The writer is also rewarded for his work.

Each participant $\mathcal{P}_i$ owns a wallet $\texttt{acc}_{\mathcal{P}_i}$ that contains an amount of useful coins related to a pair of public and secret keys $(pk_{\mathcal{P}_i}, sk_{\mathcal{P}_i})$.

8.2.2 Five building blocks

In this section, we describe the following five mechanisms that we use in our UW protocol: proof of correct computation, group selection, winner and writer election, verification process and reward distribution.

Proof of correct computation. For each executed code, a worker has to prove the correct execution of the problem's code. This ensures that the workers have invested computing power and time into the blockchain, and they may be rewarded for their work.

A first method may be to provide a hash of the executed computation. Then, the set of verifiers executes the corresponding code and computes the hash of their computations. If a threshold of verifiers obtains a hash equal to the hash provided by the worker, then the work is validated.

Proofs as proposed in [13] may also be used to prove the correct computations. Indeed, the authors of [13] present a Succinct Non-interactive ARgument of Knowledge (SNARK) to prove the correct execution of C programs. Given a program Φ in C and a time bound t to execute Φ, on any input x, it allows to prove the correct execution of Φ after a one-time setup requiring $O(|\Phi| \cdot t)$ cryptographic operations. The prover requires $O(|\Phi| \cdot t)$ cryptographic operations to generate the SNARK proof and the verifier performs $O(|x|)$ cryptographic operations to verify the proof.

We denote by `PROVE`, the algorithm run by the workers to prove the correctness of their computations. `PROVE` may take as input the problem's code Φ, the time t, the input x and the necessary keys and data according to the chosen protocol, and output a result *out* and a proof of correct computation $proof_c$.

Group selection. Our UW protocol uses a mechanism of group selection to choose the following groups: (1) a jury of voters to select the relevant problems, (2) a verifier set to verify the performed work and (3) a verifier set to check the

generated block. A participant may be selected proportionally to the amount of useful coins owned and his contribution to the blockchain, *e.g.* the number of generated blocks in the chain. Voters and verifiers have the same weight for the vote and verification, respectively, in their group.

A group selection as in [25] may be a first solution. The authors of [25] propose a group selection where the selected group represents the participants proportionally to their stake and with no minority being neither underrepresented nor overrepresented. The general idea is to begin with an empty group and add a new member over a number of iterations, following some specific rules for the candidate selection. The random secret election based on the VRF function of Algorand may also be an efficient group selection.

We define by `SELECTION` the algorithm executed by the participants at each step of a round to select (or learn his owned selection in) the group of voters and verifiers. `SELECTION` may take as input the index of the round k, a trusted random value $r_{1,k}$, the role of the current step, *i.e.* voters, verifiers of the performed work or verifiers for the generated block, and other necessary data for the chosen protocol. It outputs the selection proof of verifiers.

Winner and writer election. In our UW protocol, we use an election mechanism to elect (1) a *winner* who wins the useful coins for his performed work and (2) a *writer* who generates the next block to be added into the blockchain. The winner and the writer may be elected proportionally to the amount of useful coin owned. The writer is elected among the workers who provide a valid work for a submitted problem. The winner and the writer may be the same participant. Note that, an alternative mechanism to select a writer is required when there is no problem available. The writer may be also selected among all participants who want to be writer. We can use the SSLE protocol proposed in [15] or the random election of LEP-TSP introduced in [56].

We denote by `ELECTION`, the algorithm that elects one winner or writer among a set of participants. `ELECTION` takes as inputs the index of the round k, the trusted random value $r_{2,k}$ and other necessary data according to the chosen protocol, and outputs the winner or the writer with an eligibility proof.

Verification process. In our UW protocol, the sets of verifiers are selected to verify the computation works and the generated blocks. These processes prevent the workers to provide false computations and the writer to add conflicting data into the blockchain, *e.g.* double spending transactions. A solution to validate the works and blocks may be to collect a threshold of acknowledgments th from the verifiers. The two thirds classical threshold may guarantee security requirements, such as to be resilient to the Majority attack (as described in Section 8.4). Thus,

when a participant receives at least $th = \frac{2n_v}{3} + 1$ acknowledgments that validate a computation work (or a block), with n_v the group size, then he can consider the work (or the block) as valid (or as a part of the blockchain).

Reward distribution. When the block is considered as a part of the blockchain, the participants involved in this block are rewarded. A fixed reward RW is distributed among the winner, verifiers and writer. The winner earns rw_{worker}, each verifier who has verified the work of the winner earns $rw_{verifier}$ and the writer earns rw_{writer}. Thus, $RW = rw_{worker} + n_v \cdot rw_{verifier} + rw_{writer}$. The part of the worker rw_{worker} may be proportional to the work performed. Each verifier may earn the same amount $rw_{verifier}$.

A mechanism to punish malicious participation may also be planned. For example, if a threshold of verifiers invalidates the result of a worker, this latter may lose the coins committed for his participation. A further analysis of the reward mechanism is necessary to guarantee a fair reward for participants.

8.3 Our Useful work protocol

Our Useful Work (UW) protocol is divided in rounds where at each round k, one block B_k is added into the blockchain. A round is divided in these following eight steps: 1) Problem proposal, 2) Resources provision, 3) Relevant problems selection, 4) Problems distribution, 5) Useful work process and result announcement, 6) Work verification, 7) Winner election and 8) Writer election.

8.3.1 Problem proposal

In our UW protocol, a client $\mathcal{C}_i$ submits his problem $PB_{\mathcal{C}_i}$ with the following information and broadcasts it to the participants: $(\Phi, \alpha_c, \alpha_m, t, f_1, f_2, sign_{\mathcal{C}_i})$. A client $\mathcal{C}_i$ submits his problem under the form of a code Φ, *e.g.* in C, to be run. The client estimates the number of computations α_c, the memory necessary α_m and the time t to run Φ. The constant proposal problem fees f_1 is paid by the client who submits his problem and prevents any client to flood the network of problem. The problem storage fees $f_2 \cdot |\Phi|$ is also paid by the client to prevent computational waste in code and limit the code to only useful computation steps. Optionally, the client may provide also a certification *cert* generated by an external entity, *e.g.* laboratory or a university, to legitimate the origin of the problem. The client signs his problem proposal $sign_{\mathcal{C}_i}$ to prove that he has submitted it.

8.3.2 Resources provision

Each participant who wants to be a worker, *i.e.* running the code of a problem in order to win useful coins, has to make available his computing capacity. The worker $\mathcal{P}_i$ may commit a *stake* which is an amount of coins $\texttt{acc}_{\mathcal{P}_i}$ and the amount of memory β_m available for the computations. A worker who proposes a certain quantity of memory will obtain a problem proposal requiring an equivalent quantity of memory.

Note that a minimum amount of committed coins may be required to prevent an attacker to create several identities with low stake participation.

8.3.3 Relevant problems selection

Each problem proposal may be subject to a vote from a jury of voters to select the relevant problems. Indeed, selecting some problems may prevent clients to submit problems which may have a malicious impact for the society, *e.g.* breaking a secret key, or to propose an already submitted problem with a slight modification in order to earn easily useful coins.

The initial voters may be the creators of the blockchain. Each initial voter may have the same weight of vote, *i.e.* one voter is equivalent to one vote. Next, for each new block, a jury of voters is selected proportionally to his stake, *i.e.* a participant with a large stake has more chance to be selected in the jury of voters. The participants use the `SELECTION` algorithm to know who is selected as voter for the round k.

The role of the voters is to select the most relevant problems. The vote for a problem may be based on the number of computations, the quantity of memories and the time necessary to run the problem's code. Moreover, additional information such as a certificate *cert* signed by an external entity may also legitimate the problem. A voter $\mathcal{P}_i$ broadcasts his vote for a problem $PB_{\mathcal{C}_j}$ along with his proof of selection as voter outputted by `SELECTION`.

At the end of this step, the l problems $PB_{\mathcal{C}_1}, PB_{\mathcal{C}_2}, \ldots, PB_{\mathcal{C}_l}$ that have received the most votes are selected for the next step.

8.3.4 Problems distribution

The relevant problems $PB_{\mathcal{C}_1}, PB_{\mathcal{C}_2}, \ldots, PB_{\mathcal{C}_l}$ may be randomly distributed to the workers $\mathcal{P}_1, \ldots, \mathcal{P}_n$ who have committed their computing capacities at Resources provision step. A worker who has proposed a large quantity of memory will obtain a problem proposal requiring an equivalent quantity of memory.

The jury of voters agree on which worker has which problem. For each problem $PB_{\mathcal{C}_j}$, the voters randomly assign $PB_{\mathcal{C}_j}$ to the worker $\mathcal{P}_i$ such that $\beta_m \geq \alpha_m$ where

β_m is the quantity of memory committed by the worker $\mathcal{P}_i$ and α_m the memory estimated by the client $\mathcal{C}_j$. An assignment proof $assign_{PB_{\mathcal{C}_j},\mathcal{P}_i}$ is generated for each worker $\mathcal{P}_i$ that proves that $\mathcal{P}_i$ has to run the code of $PB_{\mathcal{C}_j}$ problem.

Note that, the value n may be larger or equal to l. In the case where $l < n$, two cases may be possible. First, only l workers may have a problem to execute and the jury does not assign a problem to the other $n-l$ workers. Another possible solution may assign one problem PB to several workers. In this case, a seed chosen for each worker is added into the code of the problem PB to obtain unique computations and a non predictable result. This may guarantee that each worker has to perform the computation works to propose a result.

8.3.5 Useful work process and result announcement

The worker $\mathcal{P}_i$ executes the corresponding code Φ of the assigned problem $P_{\mathcal{C}_j}$. Then, $\mathcal{P}_i$ computes $HASH(work_c)$ where $work_c$ is the set of computations executed.

The worker $\mathcal{P}_i$ broadcasts the following result $RES_{\mathcal{P}_i}$ to prove that he has performed the problem $PB_{\mathcal{C}_j}$: $PB_{\mathcal{C}_j}$ the problem proposal, $votes_{PB_{\mathcal{C}_j}}$ the set of votes from the jury of voters for the proposal $PB_{\mathcal{C}_j}$, $assign_{PB_{\mathcal{C}_j},\mathcal{P}_i}$ the proof of problem assignment generated in problem distribution step, $(out, proof_c)$ the final output of the code Φ and the proof of correct computation outputted by `PROVE`, $HASH(work_c)$ the hash of the computation executed $work_c$ and $sign_{\mathcal{P}_i}$ the signature of the participant $\mathcal{P}_i$ who has executed the code Φ.

The set of votes $votes_{PB_{\mathcal{C}_j}}$ enables to prove that the corresponding problem is in the list selected by the jury of voters in the Relevant problems selection. The worker $\mathcal{P}_i$ can prove his legitimacy to execute the code of the problem $PB_{\mathcal{C}_j}$ with the assignment proof $assign_{PB_{\mathcal{C}_j},\mathcal{P}_i}$. The worker $\mathcal{P}_i$ provides the values $(out, proof_c)$ outputted by `PROVE` algorithm to prove his honest and correct computational works. The hash value $HASH(work_c)$ ensures the integrity of his computation $work_c$. The worker $\mathcal{P}_i$ signs his result message $sign_{\mathcal{P}_i}$ to prove that he has run the assigned problem's code.

8.3.6 Work verification

Each participant receives a set of results $RES_{\mathcal{P}_1}, RES_{\mathcal{P}_2}, \ldots, RES_{\mathcal{P}_n}$. `SELECTION` is run to select the set of verifiers who verify the performed work for the round k.

If the participant $\mathcal{P}_i$ is a verifier, then $\mathcal{P}_i$ verifies the proof $proof_c$ of $RES_{\mathcal{P}_j}$. If $proof_c$ is correct, then $\mathcal{P}_i$ broadcasts his acknowledgement for the result $RES_{\mathcal{P}_j}$ along with the proof of selection outputted by `SELECTION`.

If there is a threshold th of acknowledgments for the same result $RES_{\mathcal{P}_j}$, then $RES_{\mathcal{P}_j}$ is considered as valid and the corresponding worker becomes a candidate for the winner election step described in the next paragraph.

8.3.7 Winner election

Among the workers who provide valid results, a winner is selected to win the useful coins for his computational work. A random selection with `ELECTION` algorithm chooses a worker as the winner to win the useful coins. The winner may be elected proportionally to the amount of useful coins owned in his wallet and his contribution to the blockchain. Note that a participant can earn useful coins when he is selected as winner, verifier of the result provided by the winner or writer, and his contribution may be measured to the number of blocks that he added into the blockchain, for example.

The winner and the verifiers who verify the winner's result are written into the blockchain as described in the writer election and validation step of the next paragraph. When the block is validated and considered as a part of the blockchain, then the writer and verifiers earn their corresponding part of the reward for his performed work and their verification, respectively.

The other workers who have not been chosen as winner and have proposed a valid result may recover their stake if they do not want to participate in the next winner election. In either case, the fees paid by clients may be sent to the workers who provide a valid result. A problem that has been assigned to a worker but has not received a valid result may be assigned to the next round by adding a seed in the problem's code. This prevents a participant to propose a result already performed.

8.3.8 Writer election and validation

The writer election step enables to select a participant as writer to generate and add a new block of transactions. The random election protocol `ELECTION` enables to choose one of them as writer and this latter wins the right to add his block into the blockchain. A new block B_k contains the following values: $(HASH(B_{k-1}), TXS, (winner, \pi_{winner}), \pi_{writer}, RW)$. The value $HASH(B_{k-1})$ where B_{k-1} is the last block added into the blockchain is the classical hash value to link the blocks of a blockchain. The set of valid transactions TXS has been verified to prevent issues such as the double spending as described in Section 8.4. The identity of the selected worker who wins useful coin and his winning proof $(winner, \pi_{winner})$ enable to write in immutable manner in the blockchain who wins the useful coin for the performed work. The value π_{winner} may contain the output of `ELECTION` during the winner election step along with the result message of the winner, the set of verifiers who verify his result and the threshold of acknowledgments. The value RW is the reward distributed among the winner, verifiers and writer. The writer proves his legitimacy with the value π_{writer} outputted by `ELECTION` during this step.

Then, a set of verifiers is selected with `SELECTION` to verify the block of the writer generated at round k. When the block B_k receives the threshold th of acknowledgments from the verifiers, B_k is considered as valid and an immutable part of the blockchain. The block B_k is then added into the blockchain along with the threshold th of acknowledgments. Finally, each involved participant receives his reward, *i.e.* the winner, the verifiers of the result provided by the winner and the writer who generates the corresponding block.

Note that, in the case where the elected writer is malicious or offline, and so does not provide the expected block, a mechanism to replace the writer is required. Another solution is to ensure a selection among the online participants until the end of this step as considered in [87, 36].

8.4 Security analysis

In this section, we discuss some well-known and new attacks, and indicate how our UW protocol prevents them.

Malicious problem proposal. A malicious client may propose a problem that is a malware or has a bad impact for the society. A client may also be a worker that proposes an already executed problem with (or not) a slight modification in order to easily win coins. We prevent these problems thanks to the relevant problems selection step that filters the problems submitted by the clients. This selection is based on the number of computations, the quantity of memory and the time necessary to run the problem's code. Additional information may be added into the problem proposal to refine the selection such as a certificate provided by a trusted party or a problem description that argue the relevant aspect of the problem.

Coalition problem. It is possible that several workers receive the same code to execute when there is less submitted problems than workers. In a set of malicious workers who receives the same problem, only one worker may perform the computational works for the assigned problem and shares her result and computations to others malicious workers. Thus, the set of malicious workers can decrease their computational effort while having the same chance to win useful coins. Our UW protocol avoids this by adding different seeds in the same code executed by several workers. Thus, the computation for a same code may be different from a worker to another and a coalition cannot provide the same result.

Work theft. A malicious participant may steal the result of a worker and convince other participants that she performed the work of the stolen result. Thus, the malicious participant may participate to the winner selection and so win useful coins

without performing any work. Our UW protocol prevents this theft by providing a proof that a specific participant receives a submitted problem to run. Indeed, in the problem distribution step, the voters distribute the relevant problem to participants who want to be workers. The agreement on who has which problem is verifiable via the assignment proof generated during the problem distribution step. Moreover, in the case where several workers receive the same problem, the different seeds added in the problem's code during the problem distribution step guarantee that each computational work is different and so a malicious worker cannot convince other participants that she has performed the work of another worker.

Fork problem. A fork in a blockchain occurs when two blocks extend the same block and compete to be in the main chain. This issue is limited in our UW protocol since in the writer election, only one writer is elected. Moreover, we ensure that only one block can reach the two thirds threshold of acknowledgements and so validate one block at each round.

Denial of Service attack. In a DoS attack, an attacker may flood the network of a huge amount of messages to make it unavailable. In our UW protocol, the attacker may be a malicious client that floods of fake or useless problems. We mitigate this issue thanks to the proposal and storage problem fees that limits a client to submit several and large codes. A DoS attack may also be run by a malicious worker. Indeed, she may create several identities in order to have several assignment problems and sent to one or several victim participants false results. We prevent this issue with requiring a minimum amount of committed coins to win the right to participate at the winner election and with a punishment mechanism for workers who provide invalid results and lose their committed coins.

Sybil attack. In a Sybil attack [39], an attacker creates several malicious participants under different identities to participate in the protocol, *e.g.* to increase her influence on the voting and verifying steps. Our UW protocol is Sybil resilient since the selection algorithm `SELECTION` is based on PoS mechanism with a minimum amount of committed stake.The amount of committed stake has to be suitable chosen to have a trade-off between motivating participants to take part in the protocol and the resilience to Sybil attacks. Moreover, an attacker splitting her currencies for her different identities cannot increase her influence on the protocol. She may succeed a Sybil attack only by investing at least as much coin as honest participants.

Majority attack. A majority attack may occur when the attacker controls more than the majority of resources in the system, *i.e.* useful coins in our UW protocol. An adversary that controls the majority of coin can make the blockchain useless by not forwarding messages or not participating in the vote and verification processes. However, with the two thirds threshold th of acknowledgments as described in Section 8.2, she cannot validate fake work or conflicting blocks while she controls less than two thirds of the total coin. We assume then, that our UW protocol works while two thirds of the total coin owned to honest participants. Moreover, in the long run of a cryptocurrency context, for example, collecting more than two thirds of total coin could be difficult to achieve if the useful coin acquires more value.

Selfish mining strategies. In a Selfish mining strategy [44], an attacker takes advantage of the usual forking to temporarily hide one or several blocks and then, she reveals her blocks at the suitable time to drop honestly generated blocks from the main chain. In this way, she can increase her ratio of blocks in the blockchain and thus also her reward compared to the reward she would obtain by following the honest protocol. In our UW protocol, a participant may earn reward being a writer, a winner or a verifier. A malicious writer cannot increase her incomes by withholding her block since a writer who does not provide her block is replaced by another. A winner is a worker who has already broadcasted his performed work and so a malicious one cannot increase incomes by hiding his computing effort. Moreover, the malicious worker cannot keep this work for later since an assigned problem that is unsolved is proposed for the next round with a seed in the code. As a verifier, an attacker cannot increase his reward since a non-participation to the verification process is not rewarded.

Nothing-at-stake attack. In a nothing-at-stake attack, an attacker may generate blocks on multiple chain of a fork and then may guarantee that one of them is chosen to be in the main chain and the attacker can recover the currencies invested in the discarded blocks. In our protocol, a worker who proposes false or empty results is detected during the Work verification phase where verifiers invalidate the result. Moreover, with the punishment mechanism a worker may lose her stake by sending false results.

Double spending attack. In a double spending attack, an attacker attempts to reuse the resources of transactions, generally by executing two transactions that spend the same currencies. For example, the attacker may issue a transaction that sends an amount of coins to a recipient. She may succeed to confirm it and then she may spend the same coins in another transaction, *i.e.* the second transaction

is also confirmed. We prevent this issue in the writer election and validation step where a group of verifiers is selected to check the block generated by the writer. If a threshold of verifiers validates the blocks, *i.e.* there is not conflicting data, then the block is added into the blockchain. The threshold of acknowledgments ensures also that only one block reached this threshold.

8.5 Variants and discussion

In this chapter, we describe at high-level a construction of our Useful Work protocol to replace the hash puzzle of the Bitcoin's PoW mechanism by useful work. Obviously, at each step of our UW protocol, several variants are possible.

In Section 8.2, we propose some possible protocols for the `SELECTION`, `ELECTION` and `PROVE` algorithms. The different steps described in Section 8.3 may also be done with other mechanisms. For example, the selection of relevant problems may be done via an external consortium of experts. The distribution of problem may be replaced by a mechanism where each participant may choose their problem to execute. The choice of protocols may be based on the security properties according to the application or the trade-off with the performance.

Note that, for each choice done in the different steps, several processes may be adapted. For example, the reward distribution may be reviewed to remain fair, *i.e.* each entity is rewarded according to his work invested into to the blockchain. Moreover, a security analysis may be necessary to prevent attacks as described in Section 8.4.

8.6 Summary on Useful Work

We presented the Useful Work protocol which is a new consensus protocol for blockchain based on the Proof-of-Stake and Proof-of-Work processes where the computing work and the memory space are dedicated to useful works. In our UW protocol, the participants compete to win useful coins. To this end, they have to run the code of problems submitted by clients. Then, a PoS-based random election chooses one of them to win coins. A participant is then elected to generate the new block of valid transactions along with the information related to the winner of useful coins. We also presented some new issues and showed that our UW protocol is resilient to these issues and the classical attacks on the consensus protocols and blockchain.

Several interesting topics are left for future works such as formal model and security analysis of our protocol, an implementation and tests to provide a security and performance trade-off, and a detailed analysis of the scalability requirement.

Chapter 9

Summary on consensus protocols for blockchain

Reaching a common agreement in distributed manner came from a pioneer problem called the Byzantine generals problem introduced by Lamport *et al.* in 1982. In this problem, a group of participants tries to agree on a single value without a trusted party. The Byzantine generals problem has been widely studied to provide solutions, referred to as consensus protocols.

From the first theoretical results, PBFT solution released in 1999 became the reference to construct consensus protocols. PBFT ensures the liveness and safety properties in partial synchrony while at most one third of participants are Byzantine. However, it has been designed for small and fixed set of participants and needs a large number of exchanged messages to reach an agreement.

Since the release of the Bitcoin paper by Nakamoto in 2008 that introduced a digital money system using a distributed ledger called blockchain, an increased interest in consensus protocols has emerged. The blockchain is a cryptographically secure chain of blocks designed to be immutable where new transactions can only be appended into the ledger after reaching a consensus. Nakamoto proposed the PoW consensus protocol to select a leader who wins the right to write a new block into the ledger. The Nakamoto PoW protocol has been designed to meet the scalability and incentivation needs of the public Bitcoin blockchain. However, the Nakamoto PoW protocol suffers from several issues such as the computing power waste, the fork problem, the centralization in big pools, the selfish mining, etc.

Thus, new consensus protocols for blockchain have been developed to prevent the Nakamoto PoW issues while leveraging the first theoretical results to solve the Byzantine generals problem. First consensus protocols have been based on a PoS mechanism such as Ouroboros and Algorand. Other consensus protocols aims to replace the PoW and PoS mechanisms by other ones, such as PoET or PoA processes. Others aim to make the computation useful, such as Primecoin. However,

these new protocols may also arise some other problems like the nothing-at-stake, the centralization to richer nodes, cloning attacks, etc.

With the increased emergence of blockchain, it is important to evaluate the trust that we can have in these technologies whose one of main challenges is the consensus protocol. Several security analysis have been published in the literature: attacks, strategies, formalization and formal proofs. The influential work of Bitcoin backbone is used as basis of security analysis in several research on consensus protocols for blockchain.

In this thesis, we studied leader election protocols that is considered in our works as particular cases of consensus protocols. Our research aims to provide results for private setting and therefore the scalability and incentivation requirements are not the primary challenges in our research. Indeed, the scalability in private setting may be managed by authentication process to participate in the protocol. Regarding the incentivation, we consider participants with common business interests such as processes with the lowest costs, and so this topic is left as a future work according to needs of use cases.

We choose to analyze the leader election construction of SSLE [11] and Algorand [57] since they seems to be one of the most relevant constructions in the state of the art of consensus protocols. SSLE elects exactly one leader per election. Whereas in Algorand that may be considered as a PLE protocol, one or several potential leaders may be selected and a rule selects one of them as leader, or no leader is elected.

These works on the leader election protocols of Algorand and SSLE outlined the following security properties: uniqueness, fairness, unpredictability, t-forward unpredictability and liveness. We select these properties because they are important to address well-known issues and attacks against consensus protocols, compared to security properties of the Bitcoin backbone model. As result, we proposed a new security model with these security properties [56]. We finally present a new leader election protocol called LEP-TSP that uses external RNG service providers to generate trusted random values. We prove that LEP-TSP meets these expected security properties while more than two third of participants are honest.

As additional work, we provide a high level description of our Useful Work protocol that make the computation power useful by solving any real world problem.

Part II

Quantum cryptanalysis of Misty schemes

Chapter 10

Introduction

Contents

In this first chapter, we introduce the context of our work and give also an overview of our contribution and the organization of Part II.

10.1 Context

As already seen in the general introduction, quantum cryptanalysis studies attack techniques assuming that an adversary has an access to quantum computers. This is an important research domain to help in designing post-quantum protocols.

Quantum cryptanalysis has received much more attention in the last past years. It is known that Grover's algorithm [59] could provide an exhaustive search among n elements in $O(\sqrt{n})$ time instead of $O(n)$. It seems that doubling the key-length of one block cipher could achieve the same security against quantum attackers. However, Kuwakado and Morii [72] introduced a new family of quantum attacks using Simon's algorithm [97] which could find the period of a periodic function in polynomial time in a quantum computer. Indeed, they describe a quantum distinguishing Chosen Plaintext Attack (CPA) on the 3-round Feistel scheme. This work has been then extended by Ito *et al.* [64] to a quantum Chosen Ciphertext Attack (CCA) distinguisher against the 4-round Feistel cipher.

Even if the most studied way to build pseudo-random permutations from random function or random permutation is the d-round Feistel construction, there exist other well-known constructions such as the Misty constructions that we analyze

in this part. We study generic attacks on Misty schemes where we assume that the internal permutations $f_1, \ldots, f_d$ are randomly chosen. The Misty construction is important from a practical point of view since it has been used as a generic construction to design Kasumi [43] algorithm that has been adopted as the standard blockcipher in the third generation mobile systems.

The plaintext message of a Misty scheme is denoted by $[L, R]$ that stands for *Left* and *Right* and the ciphertext message, after applying d rounds, is denoted by $[S, T]$. Misty L and Misty R schemes are two different variants of Misty schemes. Indeed, the first round of a Misty L scheme takes as input $[L, R]$ and it outputs $[R, R \oplus f_1(L)]$ with f_1 a secret permutation from n bits to n bits whereas the first round of a Misty R scheme takes as input $[L, R]$ and it outputs $[R \oplus f_1(L), f_1(L)]$ with f_1 a secret permutation from n bits to n bits. We also consider in this part a particular case of Misty L and Misty R constructions such that each round function f_i is defined by $f_i(x) = F_i(K_i \oplus x)$ with a public function F_i and a round secret key K_i. These constructions are named, respectively, *d-round Misty LKF scheme* and *d-round Misty RKF scheme*. To simplify the notation, the public functions F_i in each round are all denoted by F. These four variants of Misty schemes are studied in this part.

10.2 Our Contribution and Organization

In this part, we describe a non-adaptive quantum chosen plaintext attack (QCPA) against 4-round Misty L and Misty LKF schemes, and a QCPA against 3-round Misty R and Misty RKF schemes. These attacks enable to distinguish these Misty schemes from random permutations in polynomial time. We extend the quantum distinguishing attack against 3-round Misty RKF schemes to obtain a quantum key recovery attack against d-round Misty RKF schemes with complexity $\tilde{\mathcal{O}}(2^{(d-3)n/2})$. Then, we show that security of Misty L and Misty R schemes with 3 rounds differs regarding CPA attacks. The best known attack against Misty L schemes with 3 rounds has complexity 4 operations with 4 distinct messages. The best known attack against Misty R schemes has complexity $2^{n/2}$ operations with $2^{n/2}$ messages. In this part, we provide a security proof with the same bound $2^{n/2}$ which shows that the best known cryptanalysis against Misty R schemes is optimal.

Organization. In Chapter 11, we provide the main definition used in this part. Chapter 12 gives an overview of previous works and the new results provided in this part. In Chapter 13, we present our QCPA against the four variants of Misty schemes and the quantum key recovery attack on Misty RKF schemes. We provide also the security proof of Misty R schemes with 3 rounds against adaptive Chosen Plaintext attack (CPA-2). Finally, we summarize in Chapter 14.

Chapter 11

Definitions

Contents

In this chapter, we give the main algorithms and definitions used in this part.

11.1 Simon's and Grover's algorithms

In this section, we recall the results of the two quantum algorithms that we use in our quantum cryptanalysis. The full details on how the algorithms work can be found in [59, 97].

Simon's Problem. Given a Boolean function, $f : \{0,1\}^n \mapsto \{0,1\}^n$, that is observed to be invariant under some n-bit XOR period a, find a.

Simon presents a quantum algorithm [97] that provides exponential speedup and requires only $\mathcal{O}(n)$ quantum queries to find a.

Grover's problem. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and suppose that there exists a unique $x_0 \in \{0,1\}^n$ such that $f(x_0) = 1$. Given an oracle access to f, find x_0.

Grover presents a quantum algorithm [59] that requires $\mathcal{O}(2^{n/2})$ quantum queries to find x_0.

11.2 Misty constructions

In this section, we describe the four variants of Misty schemes that we study in this part. The set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$ is denoted by F_n and the set of all permutations from $\{0,1\}^n$ to $\{0,1\}^n$ is denoted by B_n. We have $B_n \subset F_n$. We denote by M^d a Misty scheme of d rounds: $f = M^d(f_1, \ldots, f_d)$, where $f_1, \ldots, f_d$ are permutations from n bits to n bits and f is a permutation from $2n$ bits to $2n$ bits.

11.2.1 Misty L scheme

Let f_1 be a permutation of B_n. Let L, R, S and T be elements in $\{0,1\}^n$. Then by definition we have:

$$M_L(f_1)([L,R]) = [S,T] \Leftrightarrow S = R \text{ and } T = R \oplus f_1(L)$$

Let $f_1, \ldots, f_d$ be d bijections of B_n. Then by definition we have:

$$M_L^d(f_1, \ldots, f_d) = M_L(f_d) \circ \ldots M_L(f_2) \circ M_L(f_1)$$

The permutation $M_L^d(f_1, \ldots, f_d)$ is called a *Misty L scheme* with d rounds. We describe in detail the equations of Misty L for the first four rounds.

$$\text{1 round :} \begin{cases} S = R \\ T = R \oplus f_1(L) = X^1 \end{cases} \qquad \text{2 rounds :} \begin{cases} S = X^1 \\ T = X^1 \oplus f_2(R) = X^2 \end{cases}$$

$$\text{3 rounds :} \begin{cases} S = X^2 \\ T = X^2 \oplus f_3(X^1) = X^3 \end{cases} \qquad \text{4 rounds :} \begin{cases} S = X^3 \\ T = X^3 \oplus f_4(X^2) = X^4 \end{cases}$$

The figure of Misty L schemes for the first round is given in Figure 11.1.

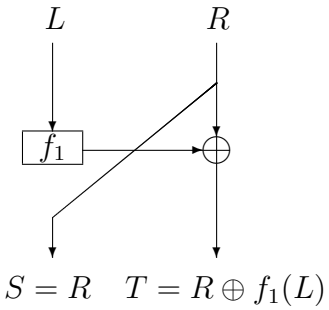


Figure 11.1: First round of Misty L

Misty LKF scheme

Let F be a public function of F_n and K_1 be a key chosen in $\{0,1\}^n$. Let L, R, S and T be elements in $\{0,1\}^n$. Then, we define:

$$M_{LKF}(F, K_1)([L, R]) = [S, T] \Leftrightarrow S = R \text{ and } T = R \oplus F(K_1 \oplus L)$$

Let $K_1, \ldots, K_d$ be d keys chosen in $\{0,1\}^n$. Then we have:

$$M^d_{LKF}(F, K_1, \ldots, K_d) = M_{LKF}(F, K_d) \circ \ldots M_{LKF}(F, K_2) \circ M_{LKF}(F, K_1)$$

In this part, we call $M^d_{LKF}(F, K_1, \ldots, K_d)$ a *Misty LKF scheme* with d rounds. The equations of the first four rounds of Misty LKF are as follows.

$$\text{1 round}: \begin{cases} S = R \\ T = R \oplus F(K_1 \oplus L) = A^1 \end{cases} \qquad \text{2 rounds}: \begin{cases} S = A^1 \\ T = A^1 \oplus F(K_2 \oplus R) = A^2 \end{cases}$$

$$\text{3 rounds}: \begin{cases} S = A^2 \\ T = A^2 \oplus F(K_3 \oplus A^1) = A^3 \end{cases} \qquad \text{4 rounds}: \begin{cases} S = A^3 \\ T = A^3 \oplus F(K_4 \oplus A^2) = A^4 \end{cases}$$

The figure of Misty LKF schemes for the first round is given in Figure 11.2.

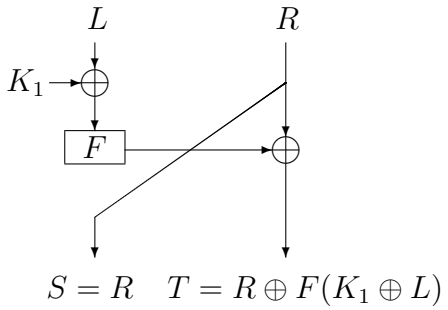


Figure 11.2: First round of Misty LKF

11.2.2 Misty R scheme

Let f_1 be a permutation of B_n. Let L, R, S and T be elements in $\{0,1\}^n$. Then by definition we have:

$$M_R(f_1)([L, R]) = [S, T] \Leftrightarrow S = R \oplus f_1(L) \text{ and } T = f_1(L)$$

Let $f_1, \ldots, f_d$ be d bijections of B_n. Then by definition we have:

$$M^d_R(f_1, \ldots, f_d) = M_R(f_d) \circ \ldots M_R(f_2) \circ M_R(f_1)$$

The permutation $M_R^d(f_1, \ldots, f_d)$ is called a *Misty R scheme* with d rounds. We describe in detail the equations of Misty R for the first four rounds.

1 round : $\begin{cases} S = R \oplus f_1(L) = Y^1 \\ T = f_1(L) \end{cases}$ 2 rounds : $\begin{cases} S = f_1(L) \oplus f_2(Y^1) = Y^2 \\ T = f_2(Y^1) \end{cases}$

3 rounds : $\begin{cases} S = f_2(Y^1) \oplus f_3(Y^2) = Y^3 \\ T = f_3(Y^2) \end{cases}$ 4 rounds : $\begin{cases} S = f_3(Y^2) \oplus f_4(Y^3) = Y^4 \\ T = f_4(Y^3) \end{cases}$

The figure of Misty R schemes for the first round is given in Figure 11.3.

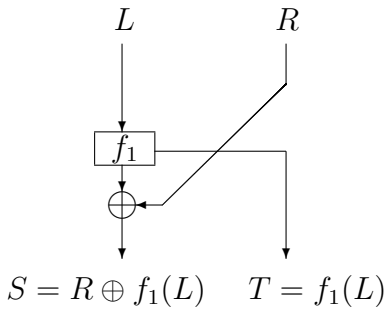


Figure 11.3: First round of Misty R

Misty RKF scheme

Let F be a public function of F_n and K_1 be a key chosen in $\{0,1\}^n$. Let L, R, S and T be elements in $\{0,1\}^n$. Then, we define:

$$M_{RKF}(F, K_1)([L, R]) = [S, T] \Leftrightarrow S = R \oplus F(K_1 \oplus L) \text{ and } T = F(K_1 \oplus L)$$

Let $K_1, \ldots, K_d$ be d keys chosen in $\{0,1\}^n$. Then we have:

$$M_{RKF}^d(F, K_1, \ldots, K_d) = M_{RKF}(F, K_d) \circ \ldots M_{RKF}(F, K_2) \circ M_{RKF}(F, K_1)$$

In this part, we call $M_{RKF}^d(F, K_1, \ldots, K_d)$ a *Misty RKF scheme* with d rounds. The equations of Misty RKF for the first four rounds are as follows:

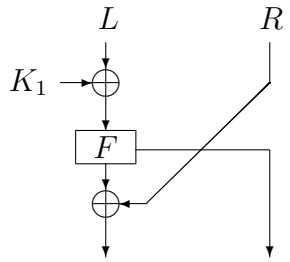
1 round : $\begin{cases} S = R \oplus F(K_1 \oplus L) = B^1 \\ T = F(K_1 \oplus L) \end{cases}$

2 rounds : $\begin{cases} S = F(K_1 \oplus L) \oplus F(K_2 \oplus B^1) = B^2 \\ T = F(K_2 \oplus B^1) \end{cases}$

3 rounds : $\begin{cases} S = F(K_2 \oplus B^1) \oplus F(K_3 \oplus B^2) = B^3 \\ T = F(K_3 \oplus B^2) \end{cases}$

4 rounds : $\begin{cases} S = F(K_3 \oplus B^2) \oplus F(K_4 \oplus B^3) = B^4 \\ T = F(K_4 \oplus B^3) \end{cases}$

The figure of Misty RKF schemes for the first round is given in Figure 11.4.



$S = R \oplus F(K_1 \oplus L) \qquad T = F(K_1 \oplus L)$

Figure 11.4: First round of Misty RKF

Chapter 12

Overview of (quantum) cryptanalysis on Misty schemes

Contents

In this chapter, we review the cryptanalysis results of the state of the art on the Misty L and Misty R schemes and we point out the new results provided in this part.

12.1 Misty L schemes with few rounds

In Table 12.1, we summarize the cryptanalysis results on few rounds of Misty L schemes based on the state of the art distinguishing attacks presented in [84, 83] together with our new contributions.

On Misty L schemes with 1 round, we have $S = R$ which gives an attack with one message in all security models. We only have to check whether S is equal to R. For a Misty L scheme, this happens with probability 1 whereas for a random permutation it happens with probability $\frac{1}{2^n}$.

On Misty L schemes with 2 rounds, we have two cases depending on the security model. For Chosen Plaintext Attack (CPA), we can choose 2 messages $[L_1, R_1]$ and $[L_2, R_2]$ such that $L_1 = L_2$. Then, we can check whether $S_1 \oplus S_2$ is equal to $R_1 \oplus R_2$. For a Misty L scheme, this happens with probability 1 whereas for a random permutation it happens with probability $\frac{1}{2^n}$. This cryptanalysis result is

	KPA	CPA	CCA	QCPA	QCCA
M_L^1	1	1	1	1	1
M_L^2	$2^{n/2}$	2	2	2	2
M_L^3	2^n	4	3	4	3
M_L^4	2^n	$2^{n/2}$	4	**Our contribution:** n (distinguishing attack)	4

Table 12.1: Number of computations to distinguish Misty L schemes (with 1, 2, 3 and 4 rounds) from random permutations

valid for other security models Chosen Ciphertext Attack (CCA), Quantum Chosen Plaintext Attack (QCPA) and Quantum Chosen Ciphertext Attack (QCCA). For Known Plaintext Attack (KPA) model, the CPA attack can be transformed into a KPA attack using $2^{n/2}$ messages and the birthday paradox bound to find a collision such that $L_i = L_j$.

On Misty L schemes with 3 rounds, there is a CPA attack with 4 messages [83] that can be transformed into a KPA attack with approximately 2^n messages and a CCA attack with 3 messages [84]. These two attacks also apply in the quantum model.

On Misty L schemes with 4 rounds, there is a CCA attack with 4 messages [84] that can be transformed into KPA attack or CPA attack. The same attacks in the quantum models hold. However, in this part we describe a QCPA attack that enables to distinguish a Misty L permutation from a random permutation using only n computations instead of $2^{n/2}$ computations.

12.2 Misty LKF with few rounds

The KPA, CPA and CCA attacks against Misty L schemes of [84, 83] can be applied on Misty LKF schemes. Therefore, we describe in Chapter 13 the QCPA attack that distinguishes a 4-round Misty LKF scheme from a random permutation using n computations.

12.3 Misty R schemes with few rounds

On Misty R schemes, the results on 1 and 2 rounds are similar to the case of Misty L schemes. On Misty R schemes with 3 rounds and with 4 rounds, the results of the KPA, CCA and QCCA attacks are similar to those of Misty L schemes since a Misty R scheme is the inverse of a Misty L scheme [84].

On Misty R schemes with 3 rounds, the best known attack has a complexity in $2^{n/2}$ computations with $2^{n/2}$ messages [84]. Luo *et al.* [78] present quantum attacks on 3-round Misty L and Misty R schemes using Simon's algorithm. In this part, we provide the security proof of Misty R schemes with 3 rounds against CPA-2 with the same bound $2^{n/2}$. We describe also a similar quantum attack on the 3-round Misty R structure that is a QCPA attack that distinguishes a Misty R scheme from a random permutation by using n computations.

Table 12.2 summarizes the cryptanalysis results that are distinguishing attacks on Misty R schemes based on [84] and our new contributions.

	KPA	CPA	CCA	QCPA	QCCA
M_R^1	1	1	1	1	1
M_R^2	$2^{n/2}$	2	2	2	2
M_R^3	2^n	**Our contribution:** $2^{n/2}$ (security proof)	3	**Our contribution:** n (distinguishing attack)	3
M_R^4	2^n	$2^{n/2}$	4	$2^{n/2}$	4

Table 12.2: Number of computations to distinguish Misty R schemes (with 1, 2, 3 and 4 rounds) from random permutations

12.4 Misty RKF schemes with few rounds

The state of the art distinguishing attacks on Misty R schemes are similar for Misty RKF schemes and are summarized in Table 12.3 together with our new contribution. In this part, we provide first a QCPA attack that distinguishes a 3-round Misty RKF scheme from a random permutation by using n computations. Then, we describe a QCPA attack that uses this quantum distinguishing attack on 3-round Misty RKF schemes to recover the keys of d-round Misty RKF schemes, for $d > 3$, in time $2^{(d-3)n/2}$.

	KPA	CPA	CCA	QCPA	QCCA
M^3_{RKF}	2^n	$2^{n/2}$	3	**Our contribution:** n (distinguishing attack)	3
M^6_{RKF}	2^{2n}	2^{2n}	2^{2n}	**Our contribution:** $2^{3n/2}$(key recovery)	2^{2n}
M^7_{RKF}	2^{4n}	2^{4n}	2^{4n}	**Our contribution:** 2^{2n}(key recovery)	2^{4n}
M^8_{RKF}	2^{4n}	2^{4n}	2^{4n}	**Our contribution:** $2^{5n/2}$(key recovery)	2^{4n}
M^9_{RKF}	2^{6n}	2^{6n}	2^{6n}	**Our contribution:** 2^{3n}(key recovery)	2^{6n}
M^{10}_{RKF}	2^{6n}	2^{6n}	2^{6n}	**Our contribution:** $2^{7n/2}$(key recovery)	2^{6n}
M^d_{RKF}, d odd $d \geq 9$	$2^{(d-3)n}$	$2^{(d-3)n}$	$2^{(d-3)n}$	**Our contribution:** $2^{(d-3)n/2}$(key recovery)	$2^{(d-3)n}$
M^d_{RKF}, d even $d \geq 8$	$2^{(d-4)n}$	$2^{(d-4)n}$	$2^{(d-4)n}$	**Our contribution:** $2^{(d-3)n/2}$(key recovery)	$2^{(d-4)n}$

Table 12.3: Number of computations to distinguish Misty RKF schemes from random permutations and number of computations to recover the keys when explicitly specified

Chapter 13

Contribution on Misty schemes

Contents

In this chapter, we present the quantum attack against the four variant of Misty scheme. As additional work, we provide also the security proof of Misty R scheme with 3 rounds.

13.1 Quantum cryptanalysis on Misty

In this section, we describe our QCPA attacks against the four variants of Misty schemes and the key recovery attack against Misty RKF schemes.

13.1.1 Quantum distinguishing attack on 4-round Misty L schemes

In this section, we describe a quantum chosen plaintext attack that distinguishes a 4-round Misty L scheme from a $2n$-bit random permutation in polynomial time. We also apply this attack on Misty LKF schemes to obtain a quantum distinguishing

attack on 4-round Misty LKF schemes.

Let $[L_1, R_1], [L_2, R_2], [L_3, R_3], [L_4, R_4]$ be four messages such that $L_1 \neq L_2$, $R_1 \neq R_2$, $L_3 = L_1$, $R_3 = R_2$, $L_4 = L_2$ and $R_1 = R_4$. As it has been shown in [83], for such four messages, we have:

$$X_1^3 \oplus X_2^3 \oplus X_3^3 \oplus X_4^3 = f_3(X_1^1) \oplus f_3(X_2^1) \oplus f_3(X_3^1) \oplus f_3(X_4^1)$$

where X_i^3 is the left half of $M_L^4([L_i, R_i])$ as denoted in Section 11.2. Then, we have:

$$\begin{aligned} X_1^3 \oplus X_2^3 \oplus X_3^3 \oplus X_4^3 &= f_3(X_1^1) \oplus f_3(X_2^1) \oplus f_3(X_3^1) \oplus f_3(X_4^1) \\ &= f_3(R_1 \oplus f_1(L_1)) \oplus f_3(R_2 \oplus f_1(L_2)) \oplus f_3(R_2 \oplus f_1(L_1)) \\ &\quad \oplus f_3(R_1 \oplus f_1(L_2)) \end{aligned}$$

We set $R_1 = x$ and we define the function

$$g(x) = f_3(x \oplus f_1(L_1)) \oplus f_3(R_2 \oplus f_1(L_2)) \oplus f_3(R_2 \oplus f_1(L_1)) \oplus f_3(x \oplus f_1(L_2))$$

We observe that we have $g(x) = g\left(x \oplus f_1(L_1) \oplus f_1(L_2)\right)$. Thus, the function g is periodic and the period is $f_1(L_1) \oplus f_1(L_2)$. Note that, this period works even if $x = R_2$. We can use the Simon's algorithm on g to get the period $s = f_1(L_1) \oplus f_1(L_2)$ in polynomial time.

In the case where g is constructed with a $2n$-bit random permutation instead of a 4-round Misty L scheme, g is not periodic with overwhelming probability. If we apply Simon's algorithm on g, the algorithm fails to find a period. Therefore, we can distinguish a 4-round Misty L scheme from a random permutation in polynomial time by using Simon's algorithm to check if g has a period.

Quantum distinguishing attack on 4-round Misty LKF schemes.

In the same way as for 4-round Misty L schemes, we have a quantum distinguishing attack on 4-round Misty LKF schemes.

Let $[L_1, R_1], [L_2, R_2], [L_3, R_3], [L_4, R_4]$ be four messages such that $L_1 \neq L_2$, $R_1 \neq R_2$, $L_3 = L_1$, $R_3 = R_2$, $L_4 = L_2$ and $R_1 = R_4$. We have also for Misty LKF:

$$\begin{aligned} A_1^3 \oplus A_2^3 \oplus A_3^3 \oplus A_4^3 &= F(K_3 \oplus A_1^1) \oplus F(K_3 \oplus A_2^1) \oplus F(K_3 \oplus A_3^1) \oplus F(K_3 \oplus A_4^1) \\ &= F(K_3 \oplus R_1 \oplus F(K_1 \oplus L_1)) \oplus F(K_3 \oplus R_2 \oplus F(K_1 \oplus L_2)) \\ &\quad \oplus F(K_3 \oplus R_2 \oplus F(K_1 \oplus L_1)) \oplus F(K_3 \oplus R_1 \oplus F(K_1 \oplus L_2)) \end{aligned}$$

where A_i^3 is the left half of $M_{LKF}^4([L_i, R_i])$ as denoted in Section 11.2. We set $R_1 = x$ and we define the function g by

$$\begin{aligned} g(x) &= F(K_3 \oplus x \oplus F(K_1 \oplus L_1)) \oplus F(K_3 \oplus R_2 \oplus F(K_1 \oplus L_2)) \\ &\quad \oplus F(K_3 \oplus R_2 \oplus F(K_1 \oplus L_1)) \oplus F(K_3 \oplus x \oplus F(K_1 \oplus L_2)) \end{aligned}$$

We observe that $g(x) = g(x \oplus F(K_1 \oplus L_1) \oplus F(K_1 \oplus L_2))$. Thus, the function g is periodic and the period is $F(K_1 \oplus L_1) \oplus F(K_1 \oplus L_2)$. We can use the Simon's algorithm on g to get the period $s = F(K_1 \oplus L_1) \oplus F(K_1 \oplus L_2)$ in polynomial time. Thus, we obtain a quantum distinguishing attack on a 4-round Misty LKF scheme by checking with the Simon's algorithm if g has a period.

13.1.2 Quantum distinguishing attack on 3-round Misty R schemes

In this section, we describe a quantum chosen plaintext attack that distinguishes a 3-round Misty R scheme from a $2n$-bit random permutation in polynomial time that is already known [78]. We also apply this attack on Misty RKF schemes to obtain a quantum distinguishing attack on 3-round Misty RKF schemes.

We consider the value $S \oplus T = f_2(Y^1) = f_2(R \oplus f_1(L))$ where $[S, T] = M_R^3([L, R])$ as described in Section 11.2. Let $[L_1, R], [L_2, R]$ be two messages such that $L_1 \neq L_2$. We set $R = x$ and we define the function

$$\begin{aligned} g(x) &= S_1 \oplus T_1 \oplus S_2 \oplus T_2 \\ &= f_2(x \oplus f_1(L_1)) \oplus f_2(x \oplus f_1(L_2)) \end{aligned}$$

where $[S_i, T_i] = M_R^3([L_i, R])$. We observe that $g(x) = g\left(x \oplus f_1(L_1) \oplus f_1(L_2)\right)$. Thus, g is a periodic function and the period is $f_1(L_1) \oplus f_1(L_2)$. We can use the Simon's algorithm on g to get the period $s = f_1(L_1) \oplus f_1(L_2)$ in polynomial time.

In the case where we apply Simon's algorithm on g that is constructed with a $2n$-bit random permutation, the algorithm fails to find a period with overwhelming probability. Thus, we can distinguish a 3-round Misty R scheme from a random permutation by checking with the Simon's algorithm if g has a period.

Quantum distinguishing attack on 3-round Misty RKF schemes.

In the same way as for 3-round Misty R schemes, we have a quantum distinguishing attack on 3-round Misty RKF schemes. We can also consider the value $S \oplus T = F(K_2 \oplus B^1) = F(K_2 \oplus R \oplus F(K_1 \oplus L))$ where $[S, T] = M_{RKF}^3([L, R])$ as described in Section 11.2. Let $[L_1, R], [L_2, R]$ be two messages such that $L_1 \neq L_2$. Thus, we set $R = x$ and we define the function g by

$$\begin{aligned} g(x) &= S_1 \oplus T_1 \oplus S_2 \oplus T_2 \\ &= F(K_2 \oplus x \oplus F(K_1 \oplus L_1)) \oplus F(K_2 \oplus x \oplus F(K_1 \oplus L_2)) \end{aligned}$$

where $[S_i, T_i] = M_{RKF}^3([L_i, R])$. We observe that $g(x) = g(x \oplus F(K_1 \oplus L_1) \oplus F(K_1 \oplus L_2))$. The function g is periodic and the period of the function is $F(K_1 \oplus$

$L_1) \oplus F(K_1 \oplus L_2)$. We can use the Simon's algorithm on g to get the period $s = F(K_1 \oplus L_1) \oplus F(K_1 \oplus L_2)$ in polynomial time.

Thus, we obtain a quantum distinguishing attack on 3-round Misty RKF schemes by using Simon's algorithm on g to check if g has a period.

13.1.3 Key recovery attack against Misty RKF schemes

Based on [62, 76, 38], we combine the quantum distinguishing attack on the 3-round Misty RKF scheme (Section 13.1.2) with the Grover search to obtain a key recovery attack against a d-round Misty RKF scheme. The attack recovers the keys of the d-round Misty RKF scheme $K_1, \ldots, K_d$. We apply the technique of [62] recalled in Proposition 1.

Proposition 1 (Proposition 3 in [62]). *Let $\Psi : F_m \times F_n \to F_n$ be a function such that $\Psi(k, \cdot) : F_n \to F_n$ is a random function for any fixed $k \in F_m$. Let $\Phi : F_m \times F_n \to F_n$ be a function such that $\Phi(k, \cdot) : F_n \to F_n$ is a random function for any fixed $k \in F_m \setminus \{k_0\}$ and $\Phi(k_0, x) = \Psi(k_0, x \oplus k_1)$. Then, given a quantum oracle access to $\Phi(\cdot, \cdot)$ and $\Psi(\cdot, \cdot)$, we can recover (k_0, k_1) with a constant probability and $\mathcal{O}((m + n^2)2^{m/2})$ queries, using $\mathcal{O}(m + n^2)$ qubits.*

For our attack, the key k_0 in Proposition 1 corresponds to the keys of the last $(d-3)$-round of a d-round Misty RKF scheme $K_4, ..., K_d$ and k_1 corresponds to the period s recovered in the quantum distinguishing attack on the 3-round Misty RKF scheme described in Section 13.1.2. The idea is to search for the correct key $k_0 = (K_4, ..., K_d)$ with the Grover search and check if $\Phi(\cdot, \cdot) \oplus \Psi(\cdot, \cdot)$ is periodic or not for the candidate key $k = (K'_4, ..., K'_d)$ by running the Simon's algorithm in parallel.

The attack is the following. Assume that we have a quantum encryption oracle of a d-round Misty RKF scheme $\mathcal{O} : \{0,1\}^{2n} \to \{0,1\}^{2n}$. For $k = (K'_4, ..., K'_d) \in \{0,1\}^{(d-3)n}$, let $D_k : \{0,1\}^{2n} \to \{0,1\}^{2n}$ denotes the partial decryption of the last $(d-3)$-round of Misty RKF with the key candidate k. Let $W : \{0,1\}^{(d-3)n} \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be the function that is the sum of the right part and the left part obtained after the 3-round of the Misty RKF scheme. W is defined by

$$W(k, L, R) := \text{the sum of the left and right halves of } D_k \circ \mathcal{O}(L, R)$$

We implement a quantum circuit of W using the quantum encryption oracle $\mathcal{O}$. In the case where $k = k_0$, then $W(k_0, L, R) = F(K_2 \oplus R \oplus F(K_1 \oplus L))$.

Then, we choose two different n-bits string α, β and define $\Psi : \{0,1\}^{(d-3)n} \times \{0,1\}^n \to \{0,1\}^n$ and $\Phi : \{0,1\}^{(d-3)n} \times \{0,1\}^n \to \{0,1\}^n$ by $\Psi(k, x) := W(k, \alpha, x)$ and $\Phi(k, x) := W(k, \beta, x)$. The function $\Psi(k, \cdot)$ is an almost random function for

each k and $\Phi(k,\cdot)$ is also an almost random function for each $k \neq k_0$. In the case where $k = k_0$, we have $\Phi(k_0, x) = \Psi(k_0, x \oplus k_1)$ where $k_1 = F(K_1 \oplus \alpha) \oplus F(K_1 \oplus \beta)$. Indeed, we have:

$$\begin{aligned}\Psi(k_0, x \oplus k_1) &= W(k, \alpha, x \oplus k_1) \\ &= F(K_2 \oplus x \oplus F(K_1 \oplus \alpha) \oplus F(K_1 \oplus \beta) \oplus F(K_1 \oplus \alpha)) \\ &= F(K_2 \oplus x \oplus F(K_1 \oplus \beta)) = W(k, \beta, x) = \Phi(k_0, x)\end{aligned}$$

Thus, we can apply Proposition 1 and recover the keys $K_4, \ldots, K_d$. Then, we can recover K_1. To this end, we construct a quantum circuit that calculates the first 3 rounds of the Misty RKF scheme. Then, we compute the period $s = F(K_1 \oplus \alpha) \oplus F(K_1 \oplus \beta)$ with the quantum distinguishing attack on the 3-round Misty RKF scheme with two arbitrary messages $[\alpha, x], [\beta, x]$ such that $x, \alpha, \beta \in \{0,1\}^n$ and $\alpha \neq \beta$. Thus, we can recover K_1 by using the Grover search. Finally, we can easily recover K_2 and K_3 using the Grover search and the recovered key K_1.

Attack complexity. By Proposition 1, we can recover $(K_4, \ldots, K_d)$ in time $\mathcal{O}(2^{(d-3)n/2})^*$. Since the last keys K_1, K_2 and K_3 are recovered by using the Grover search in time $\mathcal{O}(2^{n/2})$, the complexity of the key recovery attack against a Misty RKF scheme is $\tilde{\mathcal{O}}(2^{(d-3)n/2})$.

13.2 Security proof on 3-round Misty R

The best known CPA-1 attack against a Misty R scheme with 3 rounds is in $\mathcal{O}(2^{n/2})$ messages and computations [84]. In this section, we prove the security of the 3-round Misty R scheme against adaptive Chosen Plaintext CPA-2 attacks when the number of queries q is significantly smaller than $2^{n/2}$. Since this proof and the best known attack have the same bound $2^{n/2}$, the cryptanalysis of the 3-round Misty R scheme is optimal. For this proof, we use the result on *H coefficients technique* provided in [88].

13.2.1 H coefficient technique

Let N be a positive integer. Let I_N be the set $\{0,1\}^N$ and F_N be the set of all applications from I_N to I_N. Let B_N be the set of permutations from I_N to I_N. Let K denotes a set of k-uples of functions $(f_1, \ldots, f_k)$ of F_N. We define G as an application of $K \to F_N$.

*Taking into account the required numbers of qubits and operations, the complexity is in $\mathcal{O}(n^3 2^{(d-3)n/2})$ as explained in [62].

Definition 1 (H coefficient) *Let q be a positive integer. Let $(a_1, \ldots, a_q)$ with $a_i \in I_N$ for $i = 1, \ldots, q$ be a sequence of pairwise distinct elements of I_N . Let $(b_1, \ldots, b_q)$ with $b_i \in I_N$ for $i = 1, \ldots, q$. The H coefficient denoted by $H(a, b)$ or simply by H is the number of $(f_1, \ldots, f_k) \in K$ such that:*

$$\forall i, 1 \leq i \leq q, G(f_1, \ldots, f_k)(a_i) = b_i$$

13.2.2 Application to Misty R scheme with 3 rounds

Theorem 10 (Adaptive Chosen Plaintext attack with q queries) *[88] Let ε and β be positive real numbers. Let E be a subset of I_N^q such that $|E| \geq (1-\beta)2^{Nq}$. If for all $(a_1, \ldots, a_q)$ with $a_i \in I_N$ for $i = 1, \ldots, q$ such that $a_i \neq a_j$ when $i \neq j$ and for all $\beta \in E$ we have:*

$$H \geq \frac{|k|}{2^{Nq}}(1 - \varepsilon)$$

Then, the advantage $Adv^{\texttt{CPA-2}}$ to distinguish $G(f_1, \ldots, f_k)$ with $(f_1, \ldots, f_k) \in_R K$ from a random function $f \in_R F_N$ fulfills:

$$Adv^{\texttt{CPA-2}} \leq \beta + \varepsilon.$$

Theorem 11 (CPA-2 security on 3 rounds Misty R) *The advantage of an attacker in an adaptive chosen plaintext attack against the construction Misty R with 3 rounds is upper bounded by:*

$$Adv^{\texttt{CPA-2}} \leq \frac{3}{2}\frac{q(q-1)}{2}\frac{1}{2^n}$$

Proof. On Misty R schemes with 3 rounds, the set of keys K is equal to B_N^3 with $N = 2n$.

The transformation M_R sends $[L_i, R_i]$ to $[U_i, T_i]$ such that:

$$\begin{cases} U_i = T_i \oplus S_i = f_2(R_i \oplus f_1(L_i)) \\ T_i = f_3(f_1(L_i) \oplus U_i) \end{cases}$$

We are looking to $H = \{(f_1, f_2, f_3) \in B_n^3$ such that $\forall i, 1 \leq i \leq q, M_R[L_i, R_i] = [U_i, T_i]\}$.

Let E be the set defined as follows: $E = \{[U_i, T_i], 1 \leq i \leq q, U_i \neq U_j$ when $i \neq j\}$. We have:

$$|E| \geq 2^{Nq}\left(1 - \frac{q(q-1)}{2 \cdot 2^n}\right)$$

and we deduce that we have $\beta = \frac{q(q-1)}{2 \cdot 2^n}$.

We select f_1 such that the values $R_i \oplus f_1(L_i)$ are pairwise distinct and the values $U_i \oplus f_1(L_i)$ are pairwise distinct with $[U_i, T_i] \in E$.

- $R_i \oplus f_1(L_i) = R_j \oplus f_1(L_j)$ implies that $L_i \neq L_j$ or $R_i \neq R_j$ since $i \neq j$. Then we have to remove at most $\frac{q(q-1)}{2 \cdot 2^n}|B_n|$ permutations f_1.
- $f_1(L_i) \oplus U_i = f_1(L_j) \oplus U_j$ implies $L_i \neq L_j$ since we have $U_i \neq U_j$. Then we have to remove at most $\frac{q(q-1)}{2 \cdot 2^n}|B_n|$ permutations f_1.

Now, the function f_1 is chosen and both f_2 and f_3 are fixed in q points pairwise distinct. Then we have:

$$H \geq \frac{[B_n|^3}{2^{2nq}} \left(1 - \frac{q(q-1)}{2^n}\right) = \frac{|K|}{2^{Nq}} \left(1 - \frac{q(q-1)}{2^n}\right)$$

Then, by applying Theorem 10, we have $\varepsilon = \frac{q(q-1)}{2^n}$, $\beta = \frac{q(q-1)}{2 \cdot 2^n}$ and

$$Adv^{\mathtt{CPA-2}} \leq \left(\frac{3}{2}\right) \frac{q(q-1)}{2} \frac{1}{2^n}$$

This concludes the proof.

Chapter 14

Summary on quantum cryptanalysis of Misty schemes

Quantum and post-quantum cryptography have received considerable attention these last years from academic and industrial scientists due to the advent of quantum computers. Indeed, these computers use the quantum physic to execute tasks faster than classical computers and could make communications insecure by breaking current cryptographic mechanisms.

In 1994, Shor outlined the quantum computers threat that could break current systems based on asymmetric cryptography. Moreover, in 1998, Grover proposed an algorithm that could be a threat for symmetric cryptography. Even if today, quantum computers are not enough powerful to break current cryptographic systems, identifying new quantum attacks and designing new post-quantum protocols are important studies for being prepared to deal with this quantum threat.

Thus, in this second and last part, we provided a quantum cryptanalysis of four variants of Misty schemes [55]. These schemes are symmetric constructions used to construct Kasumi algorithm, the standard blockcipher in the third generation mobile systems. We analyzed four variants of Misty schemes, Misty L, Misty R, Misty LKF and Misty RKF.

We described QCPA attacks that enable to distinguish Misty L and Misty LKF schemes with 4 rounds, and Misty R and Misty RKF schemes with 3 rounds, from random permutations in complexity $\mathcal{O}(n)$ instead of $\mathcal{O}(2^{n/2})$. Note that the QCPA attack on 3-round Misty R schemes is already known in [78]. Moreover, we extended the quantum distinguishing attack on 3-round Misty RKF schemes to obtain a key recovery attack which recovers the keys of d-round Misty RKF schemes in time $\mathcal{O}(2^{(d-3)n/2})$. Finally, as additional work, we provided the security proof of 3-round Misty R schemes against CPA-2 attack with a complexity in $\mathcal{O}(2^{n/2})$. Since the best known attack against the 3-round Misty R schemes has the same bound, this shows that the state of the art attack is then optimal.

General conclusion

In this thesis, we studied the two following topics related to research domains become popular these last years: consensus protocols for blockchain technologies and quantum cryptanalysis of Misty schemes. For each topic, several contributions have been presented and summarize in this last chapter. Finally, we conclude with an insight of future works.

Contributions summary

In Part I, we provided research on the security of some consensus protocols for blockchain. Firstly, in Chapter 3 and Chapter 4, we presented a state of the art on consensus protocols before and since the emergence of blockchain technologies.

Reaching a common agreement without a central authority has been widely studied in distributed systems. Several theoretical results have been proved and the Practical Byzantine Fault Tolerance protocol became the reference to construct consensus protocols. The security properties achieved in these solutions are safety and liveness. However, with the advent of blockchain technologies, new requirements of scalability and incentivation due to public blockchain and issues related to the Bitcoin PoW protocol need to be taken into account. Therefore, the safety and liveness properties have been refined and completed to address these needs. The Bitcoin backbone model became thus the reference to analyze consensus protocols for blockchain. Mostly of following solutions to replace the Bitcoin PoW mechanism are analyzed in the Bitcoin backbone model to formally prove the security of consensus protocols, such as Ouroboros that is based on a PoS mechanism.

As result of this state of the art, we focused our research on the leader election process mainly used in consensus protocols for blockchain. Leader election enables participants to randomly select in distributed manner a leader among them who wins the right to generate and write the next block of transactions into the ledger. Thus, we studied the leader election process of two promising approaches to construct consensus protocols: SSLE schemes in Chapter 5 and Algorand protocol in Chapter 6. In a SSLE scheme, exactly one leader is elected whereas in the Algorand protocol, one or several potential leaders may be selected and a rule

selects one of them as leader. It is also possible in Algorand that there is not leader for some elections. For the research on SSLE [11], we revisited the original security model by adding the liveness into the model and refining the fairness and unpredictability properties. We showed also, via constructions of a random beacon, the importance of the refined fairness property that may prevent some issues. Then, we analyzed the unpredictability properties in the Algorand protocol [57] and outlined the suitable choice of the expected number of potential leaders needed to prevent predictable leaders.

As other result [56] presented in Chapter 7, we provided a security model of leader election protocol with five security properties: uniqueness, fairness, unpredictability, t-forward unpredictability and liveness. These properties address well-known issues and attacks against consensus protocols, compared to Bitcoin backbone model. Finally, we presented a new leader election protocol called LEP-TSP using external RNG service providers to generate trusted random values. LEP-TSP is intended to be used in private setting and meets the expected security properties.

As additional work provided in Chapter 8, we give a high level description of a new consensus protocol named Useful Work intended to make computation power useful. This protocol gives an insight on how we can replace the hash puzzle of Bitcoin PoW protocol by any world problem. We also presented some new issues and showed that our UW protocol is resilient to these issues and the classical attacks against consensus protocols.

Part II provided a quantum cryptanalysis on Misty schemes [55]. This aims to improve classical cryptanalysis already done on these schemes presented in Chapter 12. Then, we described in Chapter 13 four non-adaptive quantum chosen plaintext attacks against 4-round Misty L schemes, 4-round Misty LKF schemes, 3-round Misty R schemes and 3-round Misty RKF schemes. These attacks enable to distinguish these Misty schemes from random permutations in polynomial time. We extended the quantum distinguishing attack against 3-round Misty RKF schemes to obtain a quantum key recovery attack against d-round Misty RKF schemes. As additional work, we presented a security proof of 3-round Misty R schemes with the same bound $2^{n/2}$ which shows that the best known cryptanalysis against Misty R schemes with 3 rounds is optimal.

Future works

Several works may be interesting as future research topics. Regarding consensus protocols for blockchain, several mechanisms to guarantee liveness in the shuffling-based SSLE scheme are possible. For example, in the case where the leader is

detected as corrupted (because she decides to not reveal her eligibility proof, for example), another election may be planned or another leader is chosen by default. However, this may impact the other security properties satisfied by SSLE. A further analysis may be made as future work.

For the research on the Algorand protocol, the strategy against the t-forward unpredictability of Algorand may be extended in two ways. Firstly, by verifying the branch of the default value for each future election instead of only the first election as in the current strategy. Secondly, for any value of n_l to confirm the importance of a suitable choice for this parameter.

The new security model of leader election proposed in this manuscript aims to take into account the state of the art and new contributions to address well-known issues and attack against consensus protocols and also leader election protocols. Another future work could be to generalize this model in order to evaluate the security of Algorand and SSLE. Thus, an interesting work could be to give a formal comparison between these different protocols. Moreover, an improved model assuming compromised TSP providers or an evaluation of our LEP-TSP protocol are left as future works.

A formalization of our Useful Work is also a perspective of future work. Since several mechanisms are possible at each step of our protocol, a further study may be interesting to known which one is the suitable one for which security and performance goals.

Regarding the topic on quantum cryptanalysis, a similar approach can be used on other schemes such as on unbalanced Feistel schemes. Quantum distinguishing attack and quantum key recovery attack against unbalanced Feistel schemes is an interesting study by applying Simon's and Grover's algorithms.

Long résumé

Dans le monde d'aujourd'hui où les informations numériques et les communications sont devenues essentielles pour les individus, entreprises et gouvernements, il est nécessaire de garantir la confidentialité et l'intégrité de ces données échangées. La cryptologie, qui est la science permettant de concevoir des communications sécurisées, est devenue indispensable pour atteindre cet objectif de sécurité.

La cryptologie comprend deux domaines: la cryptographie qui étudie les techniques de défense et la cryptanalyse qui étudie les techniques d'attaque. Il existe deux approches en cryptographie: la cryptographie à clé secrète, aussi appelée cryptographie symétrique, où le secret est partagé par l'expéditeur et le destinataire pour chiffrer et déchiffrer les données, et la cryptographie à clé publique ou asymétrique qui utilise une paire de clés liées mathématiquement, une publique utilisée par l'expéditeur pour chiffrer les données et l'autre gardée privée par le destinataire pour déchiffrer.

Ces dernières années, deux domaines de recherche en cryptologie ont reçu une attention considérable de la part des scientifiques académiques et industriels: *les protocoles de consensus pour les technologies blockchain* dus à l'émergence des cryptomonnaies, et *la cryptanalyse quantique* due à la menace des ordinateurs quantiques. Naturellement, nos sujets de recherche se sont orientés vers ces deux domaines que nous avons étudiés séparément dans cette thèse.

La cryptographie dans les technologies blockchain

Les *technologies blockchain* sont devenues populaires en 2008 avec la publication du papier sur Bitcoin. Ces technologies sont basées sur une structure appelée blockchain. La structure de blockchain est un registre numérique ou une chaîne de blocs implémentée de manière distribuée, *i.e.* sans utiliser une autorité centrale pour maintenir le registre. De plus, la blockchain est conçue pour être immuable. En d'autres termes, dès qu'un bloc de données est écrit dans la blockchain, il ne peut plus être supprimé ni modifié. Dans le papier de Bitcoin, l'auteur (ou les auteurs) connu sous le nom de Satoshi Nakamoto présente le système de cryptomonnaie Bitcoin. Ce dernier est un système de monnaie numérique dont les transactions

entre utilisateurs sont groupées dans des blocs et enregistrées publiquement dans la blockchain.

Même si les technologies blockchain ont été en premier lieu utilisées pour développer des cryptomonnaies telles que Bitcoin et Ethereum, les deux premières cryptomonnaies créés et listées sur les sites web de capitalisation boursière, ces technologies ont démontré une approche pertinente dans plusieurs autres domaines. En effet, les transferts de cryptomonnaies généralisés à des transferts d'informations numériques peuvent être appliqués dans divers secteurs. Par exemple, dans les chaînes d'approvisionnement pour améliorer la traçabilité des produits, ou bien dans le marché de l'énergie pour faciliter les transactions d'énergie entre *prosommateurs* qui sont à la fois producteurs et consommateurs. En comparaison avec les approches actuelles, les technologies blockchain peuvent améliorer, faciliter ou même accélérer les processus de divers secteurs.

Rapidement, deux types de blockchain ont été identifiés: la publique et la privée. Dans une blockchain publique comme celle dans Bitcoin, n'importe qui à n'importe quel moment peut lire et ajouter de nouveaux blocs au registre. Cependant, cet aspect public n'est pas toujours nécessaire voire même non désiré dans certaines applications. Ainsi, la blockchain privée permet de restreindre la participation et la lecture du registre à seulement un groupe de participants authentifiés. Selon le cas d'usage, le type de blockchain doit être convenablement choisi.

Quelque soit le type de blockchain, un des principaux processus de ces technologies est le stockage sécurisé des transactions, *i.e.* les données enregistrées dans le registre n'ont pas été altérées et ne peuvent pas être modifiées ni supprimées. Ce processus est exécuté par les participants grâce à un *protocole de consensus* qui utilise des techniques cryptographiques empêchant les modifications des données du registre. Un protocole de consensus permet de parvenir à un accord commun de manière distribuée. Dans un contexte de blockchain, les participants utilisent un protocole de consensus pour parvenir à un accord sur le prochain bloc de transactions qui sera ajouté dans le registre. Généralement, ces protocoles de consensus sont basés sur un processus d'élection de *leader* qui choisit un des participants comme leader dont le rôle est de générer le nouveau bloc de transactions.

Par exemple, dans le protocole de consensus utilisé dans Bitcoin, aussi appelé Preuve de travail (PoW) (en anglais, Proof-of-Work), le leader est le premier participant à résoudre un puzzle cryptographique, connu sous le nom de *hash puzzle*. Ils doivent alors trouver un bloc dont sa valeur de hachage est inférieure à une valeur cible. Cependant, plusieurs études ont démontré d'importants problèmes dans ce premier protocole de consensus pour blockchain, tels qu'un gaspillage de ressources, une centralisation vers les participants avec le plus de puissance de calcul et bien d'autres.

En fait, concevoir des protocoles de consensus n'est pas un problème nouveau et

a été énormément étudié dans les systèmes distribués. Avec les nouveaux besoins liés à la participation ouverte et les problèmes en lien avec le protocole de Bitcoin, les travaux pour concevoir de nouveaux protocoles de consensus pour la blockchain tout en prenant en compte les premiers résultats conçus pour les systèmes distribués font l'objet de nombreuses recherches.

Au vu de la popularité croissante des technologies blockchain, la sécurité revendiquée par ces technologies doit être garantie. Dans cette thèse, nous étudions la sécurité de certains protocoles de consensus qui sont un des principaux défis dans les technologies blockchain. Plus précisément, nous voulons évaluer le niveau de confiance ainsi que les paramètres de sécurité des protocoles de consensus pour la blockchain.

La cryptanalyse quantique

Les ordinateurs quantiques sont des machines qui exploitent la physique quantique au lieu de l'électronique standard permettant ainsi d'effectuer des tâches plus rapidement que les ordinateurs classiques. Globalement, les ordinateurs classiques sont basés sur des calculs binaires dont les données sont représentées par des bits pouvant prendre deux valeurs, 0 ou 1. Alors que dans les ordinateurs quantiques, les données sont représentées par des bits quantiques, alias qubits, qui peuvent prendre soit les valeurs 0 ou 1 soit n'importe quelle superposition de 0 et 1.

La cryptographie quantique est donc la science qui utilise la mécanique quantique pour concevoir de nouveaux protocoles cryptographiques comme la distribution quantique de clé. La cryptanalyse quantique étudie alors les techniques d'attaques en utilisant les ordinateurs quantiques. Le terme de cryptographie post-quantique désigne les protocoles qui utilisent des ordinateurs classiques et résistent aux attaques utilisant les ordinateurs classiques et quantiques.

En 1994, Peter Shor met en avant la menace des ordinateurs quantiques qui pourraient casser les systèmes actuels basés sur la cryptographie asymétrique. En effet, la sécurité de ces systèmes est basée sur des problèmes supposés difficiles à résoudre avec des ordinateurs classiques tels que le calcul du logarithme discret ou la factorisation de grands nombres. En théorie, les ordinateurs quantiques pourraient résoudre efficacement ces problèmes et par conséquent casser les systèmes cryptographiques, ce qui rendrait les communications non sûres. Concernant les protocoles basés sur la cryptographie symétrique, les ordinateurs quantiques pourraient être aussi une menace mais elle serait moins importante. En effet, en 1998, Grover propose un algorithme quantique permettant de faire une recherche exhaustive parmi n élements en temps $O(\sqrt{n})$ au lieu de $O(n)$. Ce problème peut facilement être mitigé en doublant la taille de la clé.

Même si aujourd'hui, les ordinateurs quantiques qui existent ne sont pas assez puissants pour casser la cryptographie actuelle, l'étude de nouvelles attaques

quantiques et algorithmes post-quantiques est importante. Depuis 2017, l'Institut national des normes et de la technologie (NIST) (en anglais, National Institute of Standards and Technologies) a débuté une compétition pour standardiser les algorithmes post-quantiques à clé publique. Cette compétition a pour but de fournir plusieurs standards pour différentes applications dans les deux à cinq prochaines années. Au moment de la rédaction, cette compétition est toujours en cours.

Dans cette thèse, nous étudions la résistance quantique des schémas crypto--graphiques symétriques appelés Misty. Ces schémas ont été utilisés pour concevoir l'algorithme Kasumi, retenu comme le standard de chiffrement dans les systèmes mobiles de troisième génération. Plus précisément, nous proposons une cryptanalyse quantique de quatre variants des schémas de Misty.

Résumé des contributions

Cette thèse aborde séparemment les sujets de recherche en lien avec les protocoles de consensus pour les technologies blockchain dans une première partie et en lien avec la cryptographie quantique dans une seconde partie.

Dans la première partie, nous présentons nos recherches sur les protocoles de consensus pour les technologies blockchain. Nous fournissons tout d'abord un état de l'art des protocoles de consensus avant et depuis l'émergence des technologies blockchain. Ensuite, nous présentons trois contributions qui analysent la sécurité de trois protocoles. Ces recherches permettent ainsi de mettre en évidence des propriétés de sécurité qui ont pour but d'empêcher les attaques et stratégies connues. La quatrième contribution présente une description de haut niveau d'un nouveau protocole qui utilise la puissance de calcul pour des calculs utiles. Les contributions de cette première partie sont résumées dans les paragraphes suivants.

1. Revisite de l'analyse de sécurité des schémas de SSLE. Dans cette contribution, nous revisitons le modèle de sécurité des schémas de *Single Secret Leader Election (SSLE)* [11] en se focalisant sur la construction de *shuffling-based SSLE* qui est le schéma visant à être utilisé en pratique. Les schémas SSLE sélectionnent exactement un leader dont l'identité reste cachée jusqu'à ce qu'elle soit dévoilée.

Tout d'abord, nous ajoutons la propriété de *liveness* dans le modèle de sécurité d'un schéma de SSLE pour garantir une élection de leader même en présence de participants malicieux ou inactifs. La liveness est une propriété de sécurité classique des systèmes distribués qui garantit que de nouvelles données générées par les leaders soient continuellement ajoutées au système. Ensuite, nous revisitons la propriété de *fairness* qui garantit que chaque participant a la même probabilité

d'être élu. Nous rafinons églament la propriété de *unpredictability* qui signifie que les non-leaders ne peuvent pas deviner qui est leader avant qu'il ne révèle son identité. Nous guarantissons alors qu'il y a au moins deux participants non corrompus enregistrés pour l'élection et empêchons l'adversaire de deviner qui a été élu. Puis, pour chaque propriété, soit nous prouvons qu'elle est satisfaite par le schéma de shuffling-based SSLE, soit nous décrivons une stratégie pour casser cette propriété. Finalement, nous présentons deux constructions de générateurs de valeurs aléatoires pour justifier le besoin de notre nouvelle définition de fairness.

2. Les propriétés de *unpredictability* du protocole d'Algorand. Dans cette contribution, nous analysons les propriétés de *unpredictability* du protocole d'élection de leader d'Algorand [57]. Dans Algorand, un ou plusieurs leaders potentiels peuvent être élus et une règle permet d'en sélectionner un comme le leader. Il est également possible qu'il n'y ait pas de leader pour certaines élections. La propriété de *unpredictability* est importante pour empêcher des attaques comme les dénis de service ou bien les attaques de corruption. Nous montrons alors qu'Algorand satisfait cette propriété.

Ensuite, nous étendons cette propriété et définissons la propriété de *t-forward unpredictability*. Celle-ci permet de capturer le cas où même si l'adversaire est élu en tant que leader, il ne peut pas prédire les leaders des t prochaines élections. Cette propriété peut empêcher un adversaire de planifier quelles données seront ajoutées au registre au moment opportun, *e.g.* les transactions exécutées au bon moment peuvent être favorables dans le trading à haute fréquence.

Finalement, nous décrivons une stratégie pour le paramètre $n_l = 1$, le nombre de leaders potentiels attendus. Ainsi, avec cette stratégie, la probabilité que le leader soit corrompu et qu'elle prédise les t prochains leaders est de $\frac{f}{n}((\frac{f}{n})^t + (1 - (\frac{f}{n})^t)(\frac{f}{n})^t) + (1 - (\frac{f}{n})^t)\frac{1}{n-f} \cdot \frac{1}{n^t}$ au lieu de $(\frac{f}{n} + (1 - \frac{f}{n})\frac{1}{n-f})\frac{1}{n^t}$ avec n le nombre de participants dont au plus f peuvent être corrompus.

Nous mettons en avant dans cette première contribution les propriétés liées à l'*unpredictability* et l'importance du choix du paramètre n_l d'Algorand. Ce dernier pourrait être par erreur configuré à 1 par souci de simplification ou d'efficacité du protocole de consensus et doit être choisi de manière appropriée.

3. Modèle de sécurité et application à LEP-TSP. Comme autre résultat des travaux de recherche [56] sur ce premier sujet, nous définissons un modèle de sécurité pour les protocoles de Single Leader Election (SLE) avec cinq propriétés de sécurité: *uniqueness, fairness, unpredictability, t-forward unpredictability* et *liveness*. Ces propriétés ont été définies dans le modèle pour empêcher les attaques et stratégies bien connues qui visent les protocoles de consensus. La propriété de *uniqueness* signifie qu'exactement un leader est choisi à chaque élection. Cette

propriété est importante pour éviter des problèmes comme le fork ou les stratégies dites de *selfish*, puisqu'il n'est pas possible d'avoir deux leaders, ou deux blocs si nous sommes dans un contexte de blockchain, en compétition pour la même élection. Les autres propriétés ont déjà été mentionnées dans les paragraphes précédents.

Ensuite, nous proposons une construction de SLE appelée LEP-TSP (Leader Election Protocol based on Trusted Service Providers) qui est une nouvelle élection de leader basée sur des services externes de générateurs de valeurs aléatoires. Ce protocole vise à être utilisé dans des systèmes privés, tels que les blockchain privées. Nous prouvons alors que notre protocole LEP-TSP satisfait les propriétés de sécurité attendues. Plus précisément, le protocole LEP-TSP fonctionne tant que $f < \frac{n}{3}$ des n participants sont corrompus par un adversaire.

4. Useful work. Comme recherche additionelle sur ce sujet, nous présentons une description de haut niveau d'un nouveau protocole de consensus appelé *Useful work* (UW) qui a pour but d'utiliser la puissance de calcul et l'espace mémoire pour des calculs utiles. Plus précisément, au lieu de résoudre un puzzle avec une fonction de hachage comme dans Bitcoin, les participants exécutent le code de n'importe quels problèmes concrets soumis par des clients pour avoir la chance de gagner des monnaies appelées useful coin. Nous présentons également des nouveaux problèmes et montrons que notre protocole UW est résistant à ces attaques et aux attaques connues des protocoles de consensus. Cette contribution a pour but de donner un aperçu sur une nouvelle utilisation de la puissance de calcul pour la rendre utile. Ce travail peut servir de base pour construire un protocole de consensus qui permet de résoudre n'importe quel problème concret.

Dans la seconde partie sur une cryptanalyse quantique des schémas de Misty, nous présentons la contribution suivante.

5. Cryptanalyse quantique des schémas de Misty. Les schémas de Misty sont des schémas symétriques qui permettent de construire des protocoles de chiffrement par bloc, ici vus comme des générateurs de permutations pseudo-aléatoires. Nous décrivons des attaques quantiques non adaptatives à clairs choisis contre les schémas de Misty L à 4 tours, Misty LKF à 4 tours, Misty R à 3 tours et Misty RKF 3 à tours [55]. Ces attaques ont pour but de distinguer les schémas de Misty de permutations aléatoires en temps polynomial. Nous étendons ensuite l'attaque quantique du schéma de Misty RKF à 3 tours pour obtenir une attaque quantique permettant de retrouver la clé secrète des schémas de Misty RKF à d tours. Finalement, comme recherche supplémentaire, nous décrivons une preuve de sécurité des schémas de Misty R à 3 tours qui montre que la meilleure cryptanalyse connue des schémas de Misty R à 3 tours est optimale.

Publications

The works presented in Chapter 5, Chapter 6 and Chapter 7 and Part II resulted in the following publications, respectively.

- Amira Barki, Aline Gouget, Ambre Toulemonde. Revisiting security properties in Single Secret Leader Election. IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021, [11].
- Aline Gouget, Jacques Patarin, Ambre Toulemonde. Unpredictability properties in Algorand consensus protocol. IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021, [57].
- Aline Gouget, Jacques Patarin, Ambre Toulemonde. Leader election protocol based on external RNG services. 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS 2021, Paris, France, September 27-30, 2021 [56].
- Aline Gouget, Jacques Patarin, Ambre Toulemonde. (Quantum) Cryptanalysis of Misty Schemes. Information Security and Cryptology - ICISC 2020 - 23rd International Conference, Seoul, South Korea, December 2-4, 2020, [55].

The contribution of Chapter 8 has been submitted to the Blockchain And Decentralized Technologies for Social Good (BANDIT).

Bibliography

[1] Paul C. van Oorschot Alfred J. Menezes and Scott A. Vanstone. Handbook of applied cryptography, chapter hash functions and data integrity, 1996.

[2] Algorand. Algorand's official implementation in go. `https://github.com/algorand/go-algorand/blob/master/config/consensus.go`.

[3] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Commun. ACM*, 2002.

[4] Claudia Antal, Marcel Antal, Tudor Cioara, and Ionut Anghel. Trading energy as a digital asset, 05 2020.

[5] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of bitcoin transactions. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*, 2018.

[6] Sarah Azouvi and Daniele Cappelletti. Private attacks in longest chain proof-of-stake protocols with single secret leader elections. 2021.

[7] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette. *CoRR*, abs/1805.06786, 2018.

[8] Alejandro Baldominos and Yago Saez. Coin.ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning. 2019.

[9] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. *IACR Cryptol. ePrint Arch.*, 2021.

[10] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012.

[11] Amira Barki, Aline Gouget, and Ambre Toulemonde. Revisiting security properties in single secret leader election. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC, Sydney, Australia*, 2021.

[12] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. In *IEEE International Symposium on Parallel Distributed Processing*, 2009.

[13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 90–108. Springer Berlin Heidelberg, 2013.

[14] BitShares blockchain foundation. The bitshares blockchain, 2015. `https://www.bitshares.foundation/papers/BitSharesBlockchain.pdf`.

[15] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. *IACR Cryptology ePrint Archive*, 2020:25, 2020.

[16] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Advances in Cryptology - CRYPTO 2018 Proceedings, Part I*, pages 565–596, 2018.

[17] Joseph Bonneau. Why buy when you can rent? bribery attacks on bitcoin-style consensus. 2016.

[18] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. 2015.

[19] Mic Bowman, Debajyoti Das, Avradip Mandal, and Hart Montgomery. On elapsed time consensus protocols. 2021.

[20] BTC.com. Pool distribution. https://btc.com/stats/pool.

[21] Vitalik Buterin. Ethereum white paper: A next generation smart contract and decentralized application platform, 2013.

[22] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. 2014.

[23] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan.*

[24] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[25] Alfonso Cevallos and Alistair Stewart. A verifiably secure and proportional committee election rule. 2020.

[26] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Arash Pourdamghani. Hybrid mining: Exploiting blockchain's computational power for distributed problem solving. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19.

[27] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203. Plenum Press, New York, 1982.

[28] Jing Chen and Silvio Micali. Algorand. *CoRR*, abs/1607.01341, 2016.

[29] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[30] Nikos Chondros, Stathis Maneas, Christos Patsonakis, Panos Diamantopoulos, and Mema Roussopoulos. Practical asynchronous interactive consistency. 2014.

[31] Brian A. Coan and Russell Turpin. Extending binary byzantine agreement to multivalued byzantine agreement, 1984.

[32] Mauro Conti, Ankit Gangwal, and Michele Todero. Blockchain trilemma solver algorand has dilemma over undecidable messages. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA, 2019. Association for Computing Machinery.

[33] Intel Corporation. Poet 1.0 specification.

[34] Wei Dai. B-money, 1998. Accessed: 2016-04-31.

[35] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. Cryptology ePrint Archive, Report 2016/919, 2016. `https://ia.cr/2016/919`.

[36] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security*. Springer International Publishing, 2019.

[37] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[38] Xiaoyang Dong and Xiaoyun Wang. Quantum key-recovery attack on Feistel structures. *Sci. China Inf. Sci.*, 61(10):102501:1–102501:7, 2018.

[39] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*. Springer Berlin Heidelberg, 2002.

[40] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[41] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. *CoRR*, abs/1902.10244, 2019.

[42] Mourad el Maouchi, Oguzhan Ersoy, and Zekeriya Erkin. DECOUPLES: a decentralized, unlinkable and privacy-preserving traceability system for the supply chain. In Chih-Cheng Hung and George A. Papadopoulos, editors, *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 364–373. ACM, 2019.

[43] ETSI. Specification of the 3GPP Confidentiality and Integrity Algorithm KASUMI. Document available at http://www.etsi.org/.

[44] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *CoRR*, abs/1311.0243, 2013.

[45] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, page 127–140, Berlin, Heidelberg, 1983. Springer-Verlag.

[46] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[47] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. 2014. https://eprint.iacr.org/2014/765.

[48] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. 2020. https://eprint.iacr.org/2014/765.

[49] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310. Springer, 2015.

[50] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty, 2016. Accessed: 2017-02-06.

[51] Hector Garcia Molina, Frank Pittelli, and Susan Davidson. Applications of byzantine agreement in database systems. *ACM Trans. Database Syst.*, 11(1):27–47, March 1986.

[52] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.

[53] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017.

[54] Aline Gouget. Consensus protocol for permissioned ledgers, 2018-07-17.

[55] Aline Gouget, Jacques Patarin, and Ambre Toulemonde. (Quantum) Cryptanalysis of misty schemes. In *Information Security and Cryptology - ICISC 2020 - 23rd International Conference, Seoul, South Korea, December 2-4, 2020, Proceedings*, volume 12593 of *Lecture Notes in Computer Science*. Springer, 2020.

[56] Aline Gouget, Jacques Patarin, and Ambre Toulemonde. Leader election protocol based on external RNG services. In *3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS 2021, Paris, France, September 27-30, 2021*. IEEE, 2021.

[57] Aline Gouget, Jacques Patarin, and Ambre Toulemonde. Unpredictability properties in algorand consensus protocol. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021*. IEEE, 2021.

[58] Martin Grothe, Tobias Niemann, Juraj Somorovsky, and Jörg Schwenk. Breaking and fixing gridcoin. In *Proceedings of the 11th USENIX Conference on Offensive Technologies*, WOOT'17. USENIX Association, 2017.

[59] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, 1996.

[60] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, page 129–144, USA, 2015. USENIX Association.

[61] Jason Hoelscher. Diffused art and diffracted objecthood: Painting in the distributed field. 02 2014.

[62] Akinori Hosoyamada and Yu Sasaki. Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 386–403. Springer, 2018.

[63] Input Output Hong Kong IOHK. Cardano. https://whycardano.com/, 2015.

[64] Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata. Quantum Chosen-Ciphertext Attacks Against Feistel Ciphers. In Mitsuru Matsui, editor, *Topics in Cryptology - CT-RSA 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 391–411. Springer, 2019.

[65] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols(extended abstract), 1999.

[66] Vedran Juričić, Matea Radošević, and Ena Fuzul. Optimizing the resource consumption of blockchain technology in business systems. *Business Systems Research Journal*, 2020.

[67] Nikita Karandikar, Antorweep Chakravorty, and Chunming Rong. Blockchain based transaction system with fungible and non-fungible tokens for a community-based energy infrastructure. *Sensors*, 21(11), 2021.

[68] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388. Springer, 2017.

[69] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. http://primecoin.io/bin/primecoin-paper.pdf. 2013.

[70] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.

[71] Randhir Kumar and Rakesh Tripathi. Traceability of counterfeit medicine supply chain through blockchain, 01 2019.

[72] Hidenori Kuwakado and Masakatu Morii. Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In *IEEE International Symposium on Information Theory, ISIT 2010, Proceedings*, pages 2682–2685. IEEE, 2010.

[73] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.*, 3(2):125–143, March 1977.

[74] Leslie Lamport. Paxos made simple, 2001.

[75] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[76] Gregor Leander and Alexander May. Grover Meets Simon - Quantumly Attacking the FX-construction. In *ASIACRYPT*, pages 161–178. Springer, 2017.

[77] Andrei Lihu, Jincheng Du, Igor Barjaktarevic, Patrick Gerzanics, and Mark Harvilla. A proof of useful work for artificial intelligence on the blockchain. 2020.

[78] Y Y Luo, H L Yan, L Wang, H G Hu, and X J Lai. Study on block cipher structures against simon's quantum algorithm. *Journal of Cryptologic Research*, 6(5):561, 2019.

[79] Ralph C. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, 1980.

[80] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science*, pages 120–130, New York, NY, October 1999. IEEE.

[81] A Miller and LaViola JJ. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin, 2014.

[82] Tatsuo Mitani and Akira Otsuka. Traceability in permissioned blockchain. *IEEE Access*, 8:21573–21588, 2020.

[83] Valérie Nachef, Jacques Patarin, and Joana Treger. Generic Attacks on Misty Schemes -5 rounds is not enough-. *IACR Cryptology ePrint Archive*, 2009:405, 2009.

[84] Valérie Nachef, Jacques Patarin, and Joana Treger. Generic Attacks on Misty Schemes. In *Progress in Cryptology - LATINCRYPT 2010, Proceedings*, volume 6212 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2010.

[85] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[86] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. `https://ia.cr/2016/454`.

[87] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*. Springer International Publishing, 2017.

[88] Jacques Patarin. The "Coefficients H" Technique. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[89] Viktor Baranov Pavel KhahulinIgor, Igor Barinov. Poa network white paper, 2018. `https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper`.

[90] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[91] Michael O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.

[92] Juan M Roman-Belmonte, Hortensia De la Corte-Rodriguez, and E Carlos Rodriguez-Merchan. How blockchain technology can change medicine. *Postgraduate medicine*, 130(4):420—427, May 2018.

[93] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. In *28th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2021.

[94] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy, SP*. IEEE, 2020.

[95] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *In CRYPTO*, pages 148–164. Springer-Verlag, 1999.

[96] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 124–134, USA, 1994. IEEE Computer Society.

[97] Daniel R. Simon. On the Power of Quantum Computation. *SIAM J. Comput.*, 26(5):1474–1483, 1997.

[98] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy, SP*. IEEE Computer Society, 2017.

[99] Nick Szabo. Bit gold, 2005. Accessed: 2016-04-31.

[100] Yongge Wang. Another look at ALGORAND. *CoRR*, abs/1905.04463, 2019.

[101] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *CoRR*, abs/1904.04098, 2019.

[102] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview, 2018-10-03 2018.

[103] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert van Renesse. REM: resource-efficient mining for blockchains. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*.