PRIVILEGE-BASED DECENTRALIZED DATA SHARING

By

Ehab Zaghloul

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Electrical Engineering — Doctor of Philosophy

2020

ABSTRACT

PRIVILEGE-BASED DECENTRALIZED DATA SHARING

By

Ehab Zaghloul

In this dissertation, cryptographic mechanism-based data sharing schemes are presented that address the challenges of dependence on third trusted parties to facilitate sensitive data sharing and censorship. This work is driven by the lack of elevated security and privacy countermeasures necessary for use with sensitive data that many third parties exhibit. Our proposed schemes transition from centralized to distributed models, thus alleviating trust on third parties to realize data owner sharing preferences.

We first propose a secure Privilege-based Multilevel Organizational Data sharing (P-MOD) scheme that uses the cloud as a storage medium. P-MOD integrates a privilege-based access structure into an attribute-based encryption mechanism to facilitate sensitive data sharing in hierarchical settings. This structure allows data owners to share their sensitive data selectively among all levels of the hierarchy in a fine-grained manner. It also reduces computational complexity by minimizing the overall cryptographic operations.

Following the development of P-MOD, we wished to gain a better understanding of distributed systems in an effort to eliminate the need to trust third parties. Therefore, we conducted a comprehensive study of the first system to adopt blockchain, Bitcoin. In this study, we aimed to identify the security points of weakness of these distributed systems. We delved deeply into one of the major security threats, double-spending attacks, by performing two thorough probability analyses of its likelihood of success. Next, we conducted a probability of success versus profitability analysis of double-spending attacks to investigate the

trade-offs between waiting time before accepting a transaction and the profitability of these attacks.

Motivated by our study of blockchain and the underlying foundation of distributed peer-to-peer (P2P) networks, we developed a *distributed* Multilevel Attribute-based EMR management (d-MABE) scheme based on our groundwork of P-MOD. The d-MABE scheme incorporates smart contracts deployed and executed over the blockchain to ensure the data sharing preferences of the data owners are maintained. It also replaces the cloud storage with a distributed storage system that is managed by a P2P network to improve the reliability of retrieving data when requested. Using electronic medical records (EMR) as a use-case, our goal is to demonstrate the benefits of alleviating dependence on the electronic record-generating institutions and thus granting data owners (patients) control of their sensitive data in a distributed manner.

To further expand our research and reflect its applicability to a wider domain space, we proposed a blockchain-based *distributed* Coercion-Resistant and Anonymous Mobile Electronic (d-CRAME) voting scheme. The proposed scheme is secure and preserves voter privacy through secure multi-party computations performed by parties of differing allegiances. It also leverages a blockchain running smart contracts as a publicly accessible and tamper-resistant bulletin board to permanently store votes and prevent double-voting. Using voting as an application, our goal is to demonstrate the potential and feasibility of designing a distributed and remote voting scheme for large-scale elections, thus increasing voter turnout and accuracy in the decision-making process.

Copyright by
EHAB ZAGHLOUL
2020

I dedicate this work to my dear and loving parents.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Jian Ren. Being so knowledgeable and dedicated to his field, Dr. Ren was the best mentor anyone could wish to have. Our conversations were always inspiring and insightful to both my academic career and personal life. I cannot thank you enough for everything you have taught me and will always be grateful.

I would like to thank Dr. Tongtong Li, Dr. Mi Zhang, and Dr. Richard Embody for serving on my committee. Your support and feedback were a great value-added to this dissertation. I would also like to express my appreciation to Dr. Tongtong Li for providing me with guidance in my personal life and cooking us the tastiest food for Thanksgiving.

I must thank my labmates, Kai and Afifi. You were both the coolest people to ever work with and I have learned so much from both of you. I will not forget these fun times.

As for my family, I cannot thank my dear father enough for pushing me in the direction of pursuing a Ph.D. You have always believed in me and motivated me in every single possible way. My beloved mother, your constant prayers and comforting words are the reasons why I am where I am today. My beautiful sisters, Shaza and Hadeel, you are both real inspirations and role models that have given me all the strength I need.

My friends were my support system during this ride. Dana, you have helped me in every single possible way, it would take me pages to thank you, so I will leave it there. Kasstawi, you have always supported me and pushed me outside of my comfort zone to achieve the impossible. Yousry, Beltagy, and Raghda, you guys were always there and cheered me up when I needed it most, thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

1.1 Overview

Nowadays, if we choose to share sensitive data with someone, we must rely on a Trusted Third Party (TTP) to accurately and efficiently deliver the data to the correct person. This arrangement is the basis of nearly all data exchanges and rarely is it possible to circumvent the need to rely on a TTP. In a perfect world, a TTP would protect user privacy and have impenetrable security measures in place. However, this perfect scenario does not exist and thus security and privacy issues are major concerns in current data sharing systems.

In a September 2018 press release, two UMass Memorial Healthcare entities are under investigation by the Massachusetts Attorney General [1]. The lawsuit claims that two separate data breaches occurred by two medical entities where former employees improperly accessed the personal and protected health information of patients for fraudulent purposes. The lawsuit claims that the healthcare entities failed to protect patient information in-line with federal and state law. The former employees were able to access names, addresses, social security numbers, clinical information, and health insurance information. As a result of this lawsuit, the UMass Memorial Healthcare entities have agreed to a hefty $230,000 settlement and a series of remedial action steps. These steps include limiting employee access to patient information and a third-party review of their current data security policies and procedures.

In this event, the detrimental effects of unprivileged access to sensitive data are evident. More so, it is apparent that data and security breaches are not confined to random entities, but can even occur in highly respected healthcare organizations. In this scenario, the TTP failed to prevent unprivileged access to the most sensitive parts of patient records. The TTP also failed to revoke access to data when a data user (employee) was no longer privileged to access it. In a situation such as this, it is valuable to be able to control which parts of the data set are shared and with whom. For instance, the duty of a physician may warrant access to patient clinical information, but rarely would a physician need access to social security numbers. In a similar manner, a hospital billing department may need access to patient insurance information, but may not need unrestricted access to patient medical records.

A healthcare organization is a representative, real-world example of a matrix where many different professions and role duties exist. Within this complex matrix, there are also hierarchies within different branches. For example, a chief surgeon may preside over resident surgeons. When there is a complex matrix such as this, current methods to share data can fall short in terms of efficiency, privacy, and security. As mentioned previously, a TTP is used today to manage this. This places the burden of privacy protection and data security solely on the TTP and removes the power from the data owner, or patient in this case. The patient relinquishes ownership over his/her personal information, which in the wrong hands could have severe ramifications.

Efficient, secure, and private data sharing of sensitive data is essential in almost every domain today. Unfortunately, data breaches continue to rise year after year. Therefore, it is of interest to develop secure cryptographic protocols that allow proper data sharing methods. These schemes have the intention of de-risking unprivileged access to sensitive data by empowering the data owners. In this dissertation, we propose sensitive data sharing schemes that address efficiency, security, and privacy. We present our scheme by applying them to two different real-world applications that may be generalized to serve other applications similarly.

1.2 Related Work

In this section, we present the related work for both centralized and distributed models.

1.2.1 Centralized Models

Over the years, cryptographic schemes have been proposed to enhance sensitive data sharing. More recently, Attribute-Based Encryption (ABE) [2, 3] schemes have evolved to provide versatility when sharing data. These schemes integrate two types of constructs: attributes and access policies. Access policies are statements that join attributes to express which users of the system are granted access and which users are denied. ABE schemes were introduced via two different approaches: Key-Policy Attribute-Based Encryption (KP-ABE) [4, 5] and Ciphertext Policy Attribute-Based Encryption (CP-ABE) [6]. In KP-ABE, each ciphertext is labeled with a set of descriptive attributes, while each private key is integrated with an access policy. For authorized data users to decrypt the ciphertext, they must first obtain a private key from the key-issuer to use in decryption. The key-issuer integrates the access policy into the keys generated. Data users can successfully decrypt a ciphertext if the set of descriptive attributes associated with the ciphertext satisfies the access policy integrated within their private keys. KP-ABE can achieve fine-grained access control and is more flexible than Fuzzy IBE. However, the data owner must trust the key-issuer to only issue private keys to data users granted the privilege of access. This is a limitation since the data owner ultimately forfeits control over which data users are granted access. On the other hand, CP-ABE is considered to be conceptually similar to Role-Based Access Control (RBAC) [7]. It gives the data owner control over which data user is able to decrypt certain ciphertexts. This is due to the access structure being integrated by the data owner into the ciphertext during encryption. It allows the private key generated by the key-issuer to only contain the set of attributes possessed by the data user. Some CP-ABE schemes [8–23] were later introduced that can provide higher flexibility and better efficiency.

Attribute-based encryption schemes such as KP-ABE and CP-ABE serve as a better solution when data users are not ranked into a hierarchy and each is independent of one another (i.e. no relationships). Therefore, Hierarchical Attribute-Based Encryption (HABE) [24, 25] was introduced that combines the Hierarchical Identity-Based Encryption (HIBE) [26] scheme and CP-ABE [6]. HABE is able to achieve fine-grained access control in a hierarchical organization. It consists of a root master that generates and distributes parameters and keys, multiple domain masters that delegate keys to domain masters at the following levels and numerous users. In this scheme, keys are generated in the same hierarchical key generation approach as the HIBE scheme. However, this scheme becomes unsuitable for practical implementation when replicas of the same attributes are administered by other domain authorities. Synchronizing attribute administration might become a challenging issue with complex organizations that have multiple domain authorities. Examples of other hierarchical schemes were later introduced in [27–30]. As a result, File Hierarchy Ciphertext Policy Attribute-Based Encryption (FH-CP-ABE) [31] was introduced. It proposes a leveled access structure to manage a hierarchical organization that shares data of various sensitivity. A single access structure was proposed that represents both the hierarchy and the access policies of an organization. Based on the possession of certain attributes, each data user is mapped into specific transport nodes (certain levels within the hierarchy) based on the access structure that the user satisfies. The main advantage of this scheme is that it provides leveled access structures that are integrated into a single access structure. As a result, storage space is saved as only one copy of the ciphertext is needed to be shared on the cloud for all data users. However, since this scheme uses a single access structure to represent the full hierarchy, the higher levels are forced to accommodate attributes of all the levels below.

1.2.2 Distributed Models

With the continuous urge to reduce reliance on third trusted parties, distributed data sharing schemes began to evolve. In 2015, a decentralized data management scheme was introduced that facilitated access-control management over a blockchain [32]. In this system, the actual data records are stored in off-blockchain storage while pointers to these records are maintained by a key-value storage over the blockchain. This solution helps simplify the amount of data processed on the blockchain. However, the method used to define access policies in this scheme does not consider hierarchical data sharing. A user that desires to share files selectively among multiple users in a hierarchy will have to define several access policies that could become a complex problem as the number of users increases drastically. One year later, MedRec [33], the first functional electronic medical record-sharing system built on some concepts from [32] was introduced. This work builds on three Ethereum [34] smart contracts that manage authentication, confidentiality, and accountability during the data sharing process. In this system, the primary entities involved in maintaining the blockchain are the parties interested in gaining data, such as researchers and public health authorities. In return, the institutions are rewarded with access to aggregate and anonymized data. However, the success of such a system is dependent on the participation of entities that maintain the system in return for data. In addition, similar to [32], MedRec does not consider hierarchical data sharing. Finally, in 2017, another functioning electronic medical record-sharing scheme was presented to provide a secure solution using the blockchain [35]. The system uses a cloud-based storage system to store the medical records. With centralized storage, the system becomes liable to a single point of failure. In contrast to the previous systems discussed, this work builds over a permissioned blockchain, a monitored blockchain where each node involved in maintaining consensus is known.

Distributed data sharing schemes have also began to evolve in more specific-related domains such as electronic voting. Smart contracts for boardroom voting systems [36] were implemented over the Ethereum blockchain using the Open Vote Network [37]. The main advantages of this system are that it is completely decentralized, provides self-tallying and achieves end-to-end characteristics. However, the system utilizes zero-knowledge proofs which increases its overall computational costs. The majority of these cryptographic computations are implemented in smart contracts and executed over the blockchain, limiting the protocol to small-scale elections. In addition, the system is not coercion-resistant and can be manipulated by last-minute voters since they can tally the results before casting their votes. To circumvent this issue, the scheme implements an optional additional round where voters initially cast a hashed version of their votes as a commitment before casting their actual encrypted votes. In this case, although last-minute voters can still compute the election results before casting their votes, they can no longer change their selection to manipulate the results. However, this method requires additional smart contract callings and computations, imposing additional costs. Another small-scale election was also introduced in [38] that relies on expensive homomorphic encryption primitives to preserve voter privacy. Similarly, incorporating such resource-intensive cryptographic computations requires significant costs and limits the scheme to small-scale elections.

Large-scale blockchain-based voting schemes were also introduced such as the scheme presented in [39]. However, it relies heavily on a trusted central authority to facilitate ballot generation. Eligible voters are required to blind their public keys along with a digital commitment and send it to the central authority that signs the blinded message and returns them to the corresponding voters. Next, the voters unblind the signed messages to end up with signed ballots. This method is performed to prevent the central authority from linking voters to their ballots. However, since the central authority cannot identify who it is signing messages for, it may be subject to DDoS attacks. To improve on the issues imposed by using blind signatures, other schemes such as [40] and [41] were proposed that utilize homomorphic encryption alongside to ring or unlikable signatures. Aside from deploying such expensive computations to enhance voter privacy, neither scheme addresses remote voting and coercion-resistance.

1.3 Summary of Contributions

In this section, we outline our proposed data sharing schemes while presenting the motivation for our transition from centralized to distributed models. The main contributions of this research are summarized in the following subsections.

1.3.1 P-MOD: Secure Privilege-Based Multilevel Organizational Data Sharing in Cloud Computing

Sharing data in privilege-based multilevel organizations initially requires data owners to identify the sensitivity of their data they wish to share. Therefore, we begin by presenting multiple data file partitioning techniques and propose a privilege-based access structure that facilitates data sharing in hierarchical settings. We then present the proposed P-MOD scheme and formally prove its security. We show that it is secure against adaptively chosen plaintext attacks under the Decisional Bilinear Diffie-Hellman (DBDH) assumption. Next, we present a performance analysis for P-MOD and compare it to three existing schemes [6, 25, 31] that aim to achieve similar hierarchical goals. To support our performance analysis, we then implement P-MOD and conduct comprehensive simulations under various scenarios using the Census Income data set [42]. We also compare our results to simulations we have conducted for two other schemes [6, 31] under the same conditions.

1.3.2 Bitcoin and Blockchain

Here, we present an extensive analysis of Bitcoin and its underlying technology, blockchain. We provide a comprehensive explanation of the primary components of Bitcoin discussed in a sequential and logical order for the readers to comprehend. The main purpose is to cultivate the readers with the necessary background on Bitcoin to consolidate their understanding of the system. Next, We delve thoroughly into the analysis of double-spending attacks. We provide a quantitative characterization between the risk of double-spending and the number of blocks to be added to the blockchain before a transaction is accepted. We show that the probability of success of performing double-spending attacks can be modeled using two distinct probabilistic models. We also show that both models result in a similar outcome. Using these probabilistic models, we present a profitability analysis of performing double-spending attacks. Our findings are useful to both Bitcoin users and miners. Miners can obtain more insight into the mining process and potential methods to maximize their profits.

1.3.3 d-MABE: ***Distributed*** Multilevel Attribute-Based EMR Management and Applications

To eliminate the dependence on trusted third parties, such as the cloud, we expand our research by developing a novel distributed electronic medical record-sharing scheme for patients that is secure, preserves the privacy of patient data, and is efficient. Within our scheme, we also propose a distributed method for verifying medical institution staff member attributes over the blockchain before issuing them access keys. This method can help minimize trust and dependencies on key-issuers when verifying attributes of staff members. Next, we conduct comprehensive security and privacy analyses of the proposed scheme. Finally, we implement our proposed scheme using smart contracts and deploy them over the Ethereum blockchain for performance evaluation and numerical demonstration.

1.3.4 d-CRAME: ***Distributed*** Coercion-Resistant and Anonymous Mobile Electronic Voting

We develop a novel distributed, coercion-resistant, and anonymous mobile electronic voting scheme that allows voters to cast their votes in large-scale elections. Our proposed scheme builds over secure multi-party computations, allowing voters to cast their votes remotely using mobile devices such as smartphones, storing them over a blockchain permanently and irreversibly. Next, we conduct comprehensive security and privacy analyses of *d*-CRAME formally proving it is secure against adaptively chosen plaintext attacks under the Decisional Diffie-Hellman (DDH) assumption. We also introduce our Indistinguishability of Encryption Keys (IND-EK) Security Game showing that *d*-CRAME is coercion-resistant. Following that, we present a performance analysis for *d*-CRAME and compare it to the existing schemes [36,40]. Finally, we implement a desktop application and an iOS mobile application for *d*-CRAME and the corresponding smart contracts which we deploy over the Ropsten Ethereum testnet. Our empirical evaluation shows that, even when running *d*-CRAME under encryption key sizes of 4096 bits, voters can cast their votes via mobile devices in less than a minute.

1.4 Dissertation Organization

The rest of this dissertation is structured as follows. In Chapter 2, we propose the Privilege-based Multilevel Organizational Data sharing (P-MOD) scheme. Following that, in Chapter 3, we present our comprehensive research on the blockchain technology and provide our results for the probability of successful double-spending attacks versus profitability. Next, in Chapter 4, we present the *distributed* Multilevel Attribute-Based EMR (*d*-MABE) scheme. In Chapter 5, we present the *distributed* Coercion-Resistant and Anonymous Mobile Electronic (*d*-CRAME) voting scheme. Finally, in Chapter 6, we conclude this dissertation and present our future work.

Chapter 2

P-MOD: Secure Privilege-Based Multilevel Organizational Data Sharing in Cloud Computing

2.1 Introduction

A critical issue for data owners is how to efficiently and securely grant privilege level-based access rights to a set of data. Data owners are becoming more interested in selectively sharing information with data users based on different levels of granted privileges. The desire to grant level-based access results in higher computational complexity and complicates the methods in which data is shared on the cloud. Research in this field focuses on finding enhanced schemes that can securely, efficiently and intelligently share data on the cloud among users according to granted access levels.

In this chapter, we propose a secure Privilege-based Multilevel Organizational Data sharing scheme (P-MOD) that incorporates a privilege-based access structure into an attribute-based encryption mechanism. In principle, data owners encrypt parts of their data under each access policy at every level to grant access to specific data users based on their data access privileges. A data user ranked at a certain level of the hierarchy can decrypt the ciphertext (at that specific level) if and only if that data user possesses a correct set of attributes that can satisfy the access policy of that level. The data user may also decrypt the ciphertexts at the lower-levels with respect to the user's level.

The rest of this chapter is organized as follows. In Section 2.2, the problem formulation is described outlining the system model and design goals. In Section 2.3, the proposed scheme, P-MOD is introduced. Following that, in Section 2.4 we formally prove the security of P-MOD based on the hardness of the Decisional Bilinear Diffie-Hellman (DBDH) problem [43]. In Sections 2.5 and 2.6, a performance analysis our empirical results of P-MOD are conducted and compared with two other schemes [6, 31]. Finally, in Section 2.7, a conclusion is drawn to summarize the work done in this research.

2.2 Problem Formulation

Consider a data owner that possesses a data file $\mathcal{F}$ and wishes to selectively share different segments of it on the cloud among a set of data users based on certain access privileges. We assume that the data users can be ranked into a hierarchy that defines their access privileges. The challenge is to provide the data owners with an efficient, secure and privilege-based method that allows them to selectively share their data files among multiple data users while minimizing the required cloud storage space needed to store the encrypted data segments.

2.2.1 Design Goals

Based on the problem described above, we have the following design goals:

- Privilege-Based Access: Data is shared in a hierarchical manner based on user privileges. Data users with more privileges (ranked at the higher levels of the hierarchy) are granted access to more sensitive parts of $\mathcal{F}$ than those with fewer privileges (ranked at the lower-levels of the hierarchy).

- Data Confidentiality: All parts of $\mathcal{F}$ are completely protected from unprivileged data users (including the storage space). Data users are entitled to access the parts of $\mathcal{F}$ corresponding to the levels they fall in or any other parts corresponding to the levels below with respect to their own.

- Fine-grained access control: The data owner has the capability to encrypt any part of $\mathcal{F}$ using any set of descriptive attributes he/she wishes, limiting access to only authorized data users. The set of descriptive attributes is defined by the data owner at the time of encryption and can be selected from an infinite pool.

- Collusion resistant: Two or more data users at the same/different level cannot combine their private keys to gain access to any part of $\mathcal{F}$ they are not authorized to access independently.

2.2.2 System Model

The general model of privilege-based data sharing among hierarchically-ranked data users is illustrated in Fig. 2.1. The system consists of four main entities:

Data owner (DO): An individual that owns a data file and wishes to selectively share it with multiple data users based on certain desired privacy preferences.

Data users (DU): A set of hierarchically-ranked individuals $\{DU_j \in \text{DU} \mid 1 \leq i \leq \infty\}$ interested in obtaining different segments of a shared data file. Data users fall into different

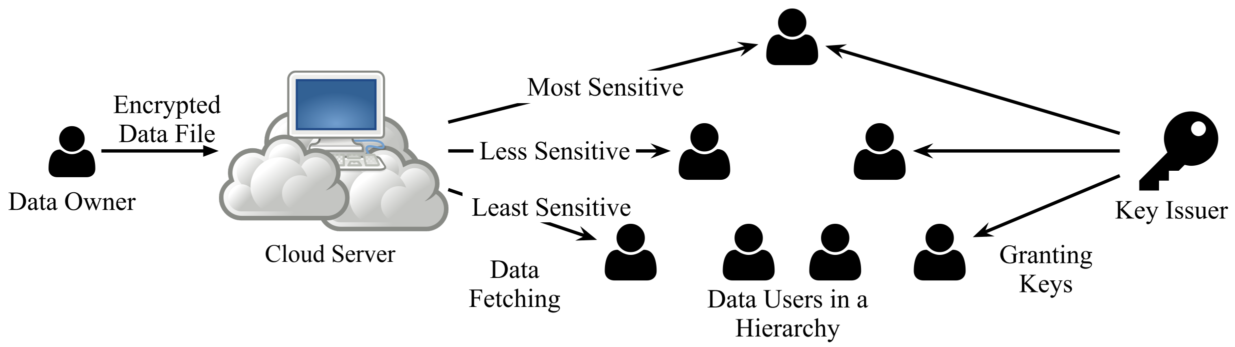


Figure 2.1: P-MOD General scheme of privilege-based data sharing.

Table 2.1: P-MOD notations summary.

Symbol	Definition
DO	A data owner
H	The hierarchical layout of the organization
$\mathcal{L}_i$	i^{th} level within H where $1 \leq i \leq k$
$\mathcal{T}_i$	i^{th} access tree at $\mathcal{L}_i$ where $1 \leq i \leq k$
F_i	i^{th} data file part where $1 \leq i \leq k$
sk_i	i^{th} symmetric key used to encrypt F_i where $1 \leq i \leq k$
EF_i	i^{th} sym. encrypted F_i under sk_i where $1 \leq i \leq k$
esk_i	i^{th} cp-abe encrypted sk_i under $\mathcal{T}_i$ at $\mathcal{L}_i$ where $1 \leq i \leq k$
DU_j	j^{th} data user in set DU where $1 \leq j \leq m$
SK_j	j^{th} data user's private key where $1 \leq j \leq m$
$A_{j,u}$	u^{th} attribute within the j^{th} data user's attribute set $\mathbb{A}_j$ where $1 \leq u \leq n$ and $1 \leq j \leq m$
x_l^i	l^{th} node of $\mathcal{T}_i$ where $1 \leq l \leq \infty$ and $1 \leq i \leq k$
$k_{x_l^i}$	Threshold value of x_l^i where $1 \leq l \leq \infty$ and $1 \leq i \leq k$

levels within the hierarchy based on specific sets of attributes $\mathbb{A}_j = \{A_{j,1}, A_{j,2} \ldots A_{j,n}\}$ they possess.

Key-issuer: A fully trusted entity that generates private keys for the data users that possess a correct set of attributes.

Cloud server: A non-trusted entity that stores the encrypted segments of the data file.

2.3 The Proposed P-MOD Scheme

We assume that file $\mathcal{F}$ is partitioned into k parts based on data sensitivity. Each part of $\mathcal{F}$ is independently encrypted and shared among the data users of the system under a privilege-based access structure.

2.3.1 Data File Partitioning and Encryption

The DO partitions file $\mathcal{F}$ into a set of k data sections, that is $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$. Each $F_i \in \mathcal{F}$ is treated as a new file that is associated with a sensitivity value used to assign access rights to the data users based on their privileges. The process of partitioning $\mathcal{F}$ is performed based on the structure of $\mathcal{F}$. We assume that $\mathcal{F}$ consists of at least one record, resulting in multiple ways to partition it as shown in Fig. 2.2.

If $\mathcal{F}$ consists of a single record, then each $F_i \in \mathcal{F}$ represents one or more record attribute(s) associated with the record, as shown in case (1) in Fig. 2.2. However, if $\mathcal{F}$ consists of multiple records, then the DO has flexibility in choosing how to partition it. One approach is to handle each record as a whole, where records are clustered into groups of similar sensitivity. In this case, each $F_i \in \mathcal{F}$ represents one or more record(s), as shown case (2) in Fig. 2.2. Alternatively, partitioning can be performed over specific record attributes, versus the whole record. In this case, $F_i \in \mathcal{F}$ represents one or more record attribute(s) of the records, as shown case (3) in Fig. 2.2.

Regardless of how partitioning is performed, each $F_i \in \mathcal{F}$ is then treated as a new data file. Suppose F_1 contains the most sensitive information of $\mathcal{F}$ that can only be accessed by a data user at the highest level $\mathcal{L}_1$ and F_k contains the least sensitive information of $\mathcal{F}$ that can be accessed by all data users at any level of the hierarchy. Before the DO

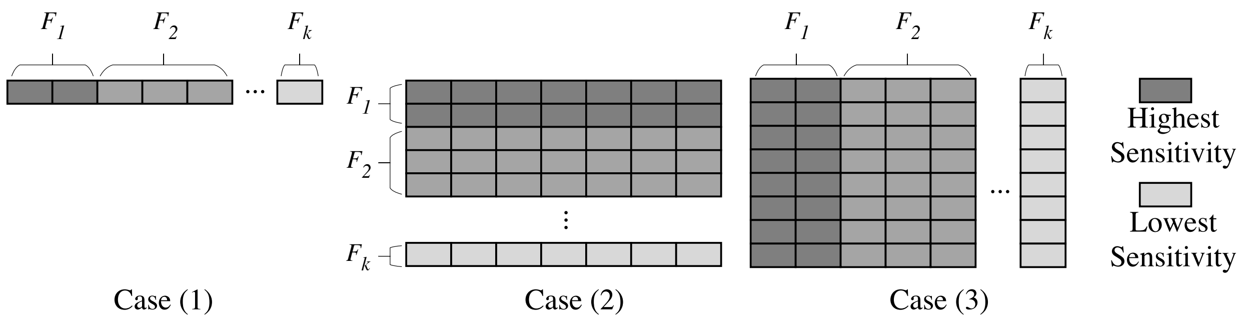


Figure 2.2: P-MOD file partitioning.

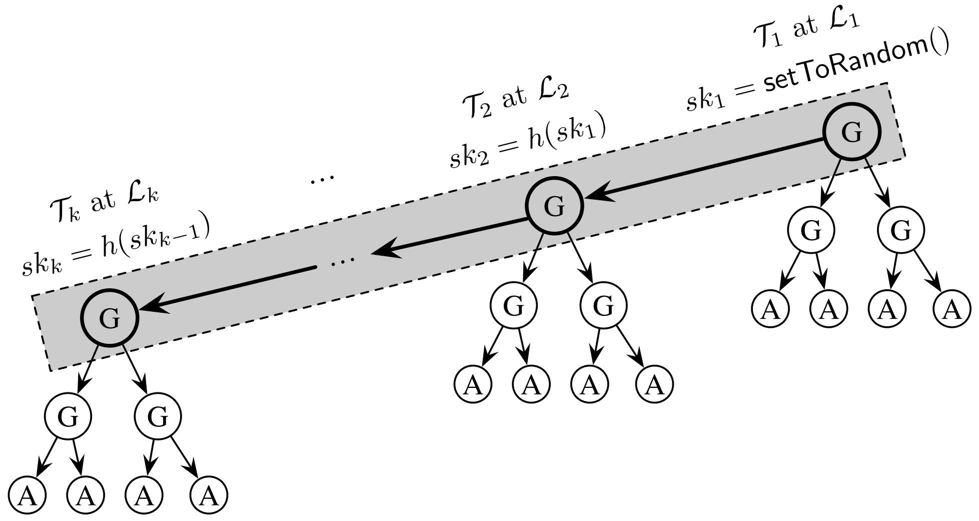
uploads $\{F_1, F_2, \ldots, F_k\}$ to the cloud, each $F_i \in \mathcal{F}$ is encrypted separately using a symmetric encryption algorithm such as the Advanced Encryption Algorithm (AES) [44] with a secret key sk_i to produce an encrypted file

$$EF_i = \mathsf{Enc}_{sk_i}(F_i). \tag{2.1}$$

For key selection, the DO randomly selects sk_1. The remaining symmetric keys $\{sk_2, \ldots, sk_k\}$ are then derived from sk_1 using a one-way cryptographic hash function h, that is

$$sk_{i+1} = h(sk_i). \tag{2.2}$$

Next, the symmetric keys are encrypted as discussed in Section 2.3.3, to be accessed only by the data users that have been granted the privilege of access. The privileged data users that are successful in obtaining sk_i corresponding to level $\mathcal{L}_i$ can derive $\{sk_{i+1}, \ldots, sk_k\}$ using equation (2.2). However, given the properties of hash function h, sk_i cannot be used to derive any of the symmetric keys $\{sk_1, \ldots, sk_{i-1}\}$.



$\mathcal{T}_1$ at $\mathcal{L}_1$
$sk_1 = \mathsf{setToRandom}()$
$\mathcal{T}_2$ at $\mathcal{L}_2$
$sk_2 = h(sk_1)$
$\mathcal{T}_k$ at $\mathcal{L}_k$
$sk_k = h(sk_{k-1})$
...
G
A

Figure 2.3: P-MOD privilege-based multilevel access structure.

2.3.2 The P-MOD Privilege-Based Access Structure

The general privilege-based access structure of P-MOD is illustrated in Fig. 2.3. Data users are ranked into k levels of privileges, $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k\}$. The DO defines an access tree $\mathcal{T}_i$ at each corresponding level $\mathcal{L}_i$. Each $\mathcal{T}_i$ is associated with the appropriate leaf nodes (attributes) that define the privileges of the level. Data users that possess the correct sets of attributes that can satisfy $\mathcal{T}_i$ at $\mathcal{L}_i$ are granted access to symmetric key sk_i, hence, can derive $\{sk_{i+1}, \ldots, sk_k\}$ and are able to decrypt parts $\{EF_i, EF_{i+1}, \ldots, EF_k\}$.

As shown in Fig. 2.3, an access tree $\mathcal{T}_i$ may consist of non-leaf nodes and leaf nodes. The non-leaf nodes are threshold gates, represented as 'G', while the leaf nodes are attributes, represented as 'A'. The DO may construct access trees from any number and layout of nodes that satisfy the privacy preferences desired.

One of the advantages of our proposed privilege-based access structure is the ability to reduce attribute replication when defining the hierarchy. Data users that possess attributes that can satisfy access tree $\mathcal{T}_i$ are granted access to sk_i, however, they do not need to possess attributes that can also satisfy the access trees $\{\mathcal{T}_{i+1}, \ldots, \mathcal{T}_k\}$ in order to obtain $\{sk_{i+1}, \ldots, sk_k\}$. This helps simplify the process of defining a hierarchy as the number of users or access constraints grow. With reduced-size access trees, we can greatly reduce the computational complexity when encrypting file partitions, generating private keys for privileged users and decrypting ciphertexts.

2.3.3 The Proposed P-MOD Construction

The scheme is based on the construction presented in [6] and formally divides the process into four main functions:

$\mathsf{Setup}(1^\kappa)$**:** This is a probabilistic function carried out by the key-issuer. The Setup function takes a security parameter κ and randomly chooses values $\alpha, \beta \in \mathbb{Z}_p$. The outputs of this function are public key PK and master key MK defined as:

$$\begin{aligned} PK &= \{\mathbb{G}_0, g, B = g^\beta, e(g,g)^\alpha\}, && (2.3) \\ MK &= \{\beta, g^\alpha\}. && (2.4) \end{aligned}$$

$\mathsf{KeyGen}(MK, \mathbb{A}_j)$**:** This is a probabilistic function carried out by the key-issuer. The inputs to this function are MK generated by the Setup function, and the attribute set $\mathbb{A}_j$ of j^{th} data user, where $A_{j,u} \in \mathbb{A}_j$ represents the u^{th} attribute within the set. The KeyGen function outputs a unique private key SK_j for the data user. In order to guarantee a unique SK_j, it generates a random value $r_j \in \mathbb{Z}_p$ and incorporates it within the private key. Based on the number of attributes in the input set $\mathbb{A}_j$, the KeyGen function also generates a random value $r_{j,u} \in_r \mathbb{Z}_p$ for each attribute within the set. The SK_j is defined as:

$$\begin{aligned} SK_j = \big(D_j = g^{(\alpha + r_j)/\beta}, \\ \{D_{j,u} = g^{r_j} \cdot h(u)^{r_{j,u}}, D'_{j,u} = g^{r_{j,u}} \mid \forall A_{j,u} \in \mathbb{A}_j\}\big). \end{aligned} \tag{2.5}$$

The purpose of the randomly selected r_j is to ensure that each SK_j is unique and the attribute components within the SK_j are associated. It should be infeasible for data users to collude by combining components of their private keys ($D_{j,u}$ and $D'_{j,u}$) to decrypt data beyond their individual access rights. This means, the attribute components from different private keys cannot be combined to access unauthorized data.

Encrypt(PK,sk_i,$\mathcal{T}_i$): This is a probabilistic function carried out by the DO to encrypt the symmetric keys that are to be shared with the privileged data users. The inputs to this function are PK, the public key generated by the Setup function, the symmetric key sk_i derived in equation (2.2) representing the data that will be encrypted, and the access tree $\mathcal{T}_i$ that defines the authorized set of attributes at $\mathcal{L}_i$. The output of this function is the encrypted symmetric key esk_i.

For $\{sk_1, \dots, sk_k\}$, the Encrypt function will run k times, once for each sk_i. At each run, the Encrypt function chooses a polynomial $q_{x_l^i}$ with degree $d_{x_l^i} = k_{x_l^i} - 1$ for each node $x_l^i \in \mathcal{T}_i$. The process of assigning polynomials to each x_l^i occurs in a top-bottom approach starting from the root node in $\mathcal{T}_i$. The Encrypt function chooses a secret $s_i \in \mathbb{Z}_p$ and sets the value of $q_{x_1^i}(0) = s_i$. Next, it randomly chooses the remaining points of the polynomial to completely define it. For any other node $x_l^i \in \mathcal{T}_i$, the Encrypt function sets the value $q_{x_l^i}(0) = q_{\text{parent}}(\text{index}(x_l^i))$, where q_{parent} is the parent node polynomial of x_l^i. The remaining points of those polynomials are then randomly chosen.

Let X_i be the set of leaf nodes in $\mathcal{T}_i$. The encrypted symmetric key esk_i at $\mathcal{L}_i$ is then constructed as:

$$\begin{aligned} esk_i = \big(\mathcal{T}_i, \tilde{C}_i = sk_i \cdot e(g,g)^{\alpha \cdot s_i}, C_i = g^{\beta \cdot s_i}, \\ \{C_{x_l^i} = g^{q_{x_l^i}(0)}, C'_{x_l^i} = h(x_l^i)^{q_{x_l^i}(0)} \mid \forall x_l^i \in X_i\}\big). \end{aligned} \tag{2.6}$$

$\mathsf{Decrypt}(esk_i, SK_j)$**:** This is a deterministic function carried out by the data user. The inputs to this function are an encrypted symmetric key esk_i corresponding to $\mathcal{L}_i$, and the private key SK_j of the j^{th} data user.

The $\mathsf{Decrypt}$ function operates in a recursive manner propagating through the nodes in $\mathcal{T}_i$ by calling a recursive function defined as $\mathsf{DecryptNode}(esk_i, SK_j, x_l^i)$. The inputs to this function are esk_i, SK_j and a node x_l^i within $\mathcal{T}_i$. If the attributes incorporated within SK_j satisfy the rules within $\mathcal{T}_i$, the data user can decrypt esk_i and obtain sk_i.

$\mathsf{DecryptNode}(esk_i, SK_j, x_l^i)$ performs differently depending on whether x_l^i is a leaf or non-leaf node. If x_l^i is a leaf node and $\mathrm{att}(x_l^i) \notin \mathbb{A}_j$, $\mathsf{DecryptNode}(esk_i, SK_j, x_l^i)$ returns $\emptyset$, otherwise, $\mathrm{att}(x_l^i) = A_{j,u} \in \mathbb{A}_j$ and $\mathsf{DecryptNode}(esk_i, SK_j, x_l^i)$ is defined as:

$$\begin{aligned} \mathsf{DecryptNode}(esk_i, SK_j, x_l^i) &= \frac{e(D_{j,u}, C_{x_l^i})}{e(D'_{j,u}, C'_{x_l^i})} \\ &= \frac{e(g^{r_j} \cdot h(u)^{r_{j,u}}, g^{q_{x_l^i}(0)})}{e(g^{r_{j,u}}, h(x_l^i)^{q_{x_l^i}(0)})} \\ &= e(g,g)^{r_j \cdot q_{x_l^i}(0)}. \end{aligned} \tag{2.7}$$

If x_l^i is a non-leaf node, $\mathsf{DecryptNode}(esk_i, SK_j, x_l^i)$ operates recursively. For each node $z_{l,c}^i$ that is a child of x_l^i, $\mathsf{DecryptNode}(esk_i, SK_j, z_{l,c}^i)$ is computed and the output is stored in $F_{z_{l,c}^i}$.

This recursive function is based on Lagrange interpolation. The Lagrange coefficient $\Delta_{a,\mathbb{A}_j}$ for $a \in \mathbb{Z}_p$ and the set of attributes $\mathbb{A}_j$ is defined as:

$$\Delta_{a,\mathbb{A}_j}(x) = \prod_{k \in \mathbb{A}_j, k \neq a} \frac{x-k}{a-k}. \tag{2.8}$$

Let $\mathbb{A}_{j,x_l^i}$ be an arbitrary $k_{x_l^i}$-sized set of child nodes $z_{l,c}^i$ such that $F_{z_{l,c}^i} \neq \emptyset$. If no such set exists then the function returns $F_{z_{l,c}^i} = \emptyset$. Otherwise, $F_{z_{l,c}^i}$ is computed using Lagrange interpolation as follows:

$$\begin{aligned}
F_{x_l^i} &= \prod_{z_{l,c}^i \in \mathbb{A}_{j,x_l^i}} F_{z_{l,c}^i}^{\Delta_{a,\mathbb{A}'_{j,x_l^i}}(0)} \\
&= \prod_{z_{l,c}^i \in \mathbb{A}_{j,x_l^i}} \left(e(g,g)^{r_j \cdot q_{z_{l,c}^i}(0)} \right)^{\Delta_{a,\mathbb{A}'_{j,x_l^i}}(0)} \\
&= \prod_{z_{l,c}^i \in \mathbb{A}_{j,x_l^i}} \left(e(g,g)^{r_j \cdot q_{\mathrm{parent(z_{l,c}^i)}}(\mathrm{index(z_{l,c}^i)})} \right)^{\Delta_{a,\mathbb{A}'_{j,x_l^i}}(0)} \qquad (2.9) \\
&= \prod_{z_{l,c}^i \in \mathbb{A}_{j,x_l^i}} e(g,g)^{r_j \cdot q_{x_l^i}(i) \cdot \Delta_{a,\mathbb{A}'_{j,x_l^i}}(0)} \\
&= e(g,g)^{r_j \cdot q_{x_l^i}(0)},
\end{aligned}$$

where $a = \mathrm{index(z_{l,c}^i)}$ and $\mathbb{A}'_{j,x_l^i} = \{\mathrm{index}(z_{l,c}^i) \colon z_{l,c}^i \in \mathbb{A}_{j,x_l^i}\}$.

If the attributes in $\mathbb{A}_j$ satisfy $\mathcal{T}_i$, then the following is computed at the root node x_1^i.

$$\begin{aligned} R_i &= \mathsf{DecryptNode}(esk_i, SK_j, x_1^i) \\ &= e(g,g)^{r_j \cdot q_{x_1^i}(0)} \\ &= e(g,g)^{r_j \cdot s_i}. \end{aligned} \tag{2.10}$$

To obtain sk_i from the result derived in equation (2.10), we compute the following:

$$\frac{\tilde{C}_i}{\frac{e(C_i, D_j)}{R_i}} = \frac{sk_i \cdot e(g,g)^{\alpha \cdot s_i}}{\frac{e\left(g^{\beta \cdot s_i}, g^{(\alpha + r_j)/\beta}\right)}{e(g,g)^{r_j \cdot s_i}}} = sk_i. \tag{2.11}$$

At this point, the data user can simply decrypt EF_i using the sk_i derived from equation (2.11) to obtain the plaintext F_i as follows:

$$F_i = \mathsf{Dec}_{sk_i}(EF_i). \tag{2.12}$$

If the data user is interested in attaining data files $\{F_{i+1} \ldots F_k\}$ belonging to levels $\{\mathcal{L}_{i+1} \ldots \mathcal{L}_k\}$ respectively, the user can compute the symmetric keys $\{sk_{i+1}, \ldots, sk_k\}$ of the lower-level using the derived sk_i as previously discussed in equation (2.2). Finally, any $\{EF_{i+1} \ldots EF_k\}$ at the lower-levels can be decrypted as in equation (2.12).

2.4 Security Analysis

In this section, a formal proof of security for P-MOD is presented. It is assumed that a symmetric encryption technique such as AES is used to secure each data file $F_i \in \mathcal{F}$. It is also assumed that the process of attribute authentication between a data user and the key-issuer, in order for the data user to obtain a private key, is secure and efficient.

Theorem 1. P-MOD is secure against unprivileged accesses assuming the hash function is collusion resistant.

Proof. Let $\mathbb{A}_j = \{A_{j,1}, A_{j,2} \dots A_{j,n}\}$ be a set of attributes possessed by user DU_j and $\mathcal{T}_i$ an access tree at level $\mathcal{L}_i$ of a hierarchy. If $\mathbb{A}_j \in \mathcal{T}_i$, the user can obtain sk_i as discussed in equations (2.7-2.11). Given hash function h, the user cannot derive $\{sk_1 \dots sk_{i-1}\}$. Therefore, P-MOD is immune to unprivileged accesses. $\square$

Following the work presented in [43], we also provide a security proof based on ciphertext indistinguishability that proves that the adversary is not able to distinguish pairs of ciphertexts. A cryptosystem is considered to be secure under this property if the probability of an adversary to identify a data file that has been randomly selected from a two-element data file chosen by the adversary and encrypted does not significantly exceed $\frac{1}{2}$. We first present an Indistinguishability under Chosen-Plaintext Attack (IND-CPA) security game [45]. Next, based on the IND-CPA security game, a formal proof of security is provided for P-MOD.

IND-CPA is a game used to test for the security of asymmetric key encryption algorithms. In this game, the adversary is modeled as a probabilistic polynomial-time algorithm. The algorithms in the game must be completed and the results returned within a polynomial number of time steps. The adversary will choose to be challenged on an encryption under a leveled access tree $\mathcal{T}^*$. The adversary can impersonate any data user and request many private keys SK_j. However, the game rules require that any attribute set $\mathbb{A}_j$ that the adversary claims to possess does not satisfy $\mathcal{T}^*$. The security game is divided into the following steps:

Initialization: The adversary selects an access tree $\mathcal{T}^*$ to be challenged against and commits to it.

Setup: The challenger runs the Setup function and sends the public key PK to the adversary.

Phase 1: The adversary requests multiple private keys $(SK_1, \ldots, SK_{q_1})$ corresponding to q_1 different sets of attributes $(\mathbb{A}_1, \ldots, \mathbb{A}_{q_1})$.

Challenge: The adversary submits two equal length data files F_0 and F_1 to the challenger. The adversary also sends $\mathcal{T}^*$ such that none of $(SK_1, \ldots, SK_{q_1})$ generated from Phase 1 contain correct sets of attributes that satisfy it. The challenger flips a coin μ randomly and encrypts F_μ under $\mathcal{T}^*$. Finally, the challenger sends the ciphertext CT^* generated according to equation (2.6) to the adversary.

Phase 2: Repeat phase 1 with the restriction that none of the newly generated private keys $(SK_{q_1+1}, \ldots, SK_q)$ corresponding to the different sets of attributes $(\mathbb{A}_{q_1+1}, \ldots, \mathbb{A}_q)$ contain correct sets of attributes that satisfy $\mathcal{T}^*$.

Guess: The adversary outputs a guess μ' of μ. The adversary wins the security game if $\mu' = \mu$ and loses otherwise.

Definition 1 (Secure against adaptively chosen plaintext attack.)**.** P-MOD is said to be secure against an adaptively chosen plaintext attack if any polynomial-time adversary has only a negligible advantage in the security game, where the advantage is defined as $Adv = Pr[\mu' = \mu] - \frac{1}{2}$.

The security of P-MOD is reduced to the hardness of the DBDH problem. Based on Theorem 1 and by proving that a single esk_i at any $\mathcal{L}_i$ is secure, the whole system is proved to be secure since all ciphertexts at any level follow the same rules.

Theorem 2. P-MOD is secure against adaptively chosen plaintext attack if the DBDH assumption holds.

Proof. Assume there is an adversary that has non-negligible advantage $\varepsilon = \mathsf{Adv}_{\mathcal{A}}$. We construct a simulator that can distinguish a DBDH element from a random element with advantage ε. Let $e \colon \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ be an efficiently computable bilinear map and $\mathbb{G}_0$ is of prime order p with generator g. The DBDH challenger begins by selecting the random parameters: $a, b, c \in_r \mathbb{Z}_p$. Let $g \in \mathbb{G}_0$ be a generator and T is defined as $T = e(g, g)^{abc}$ if $\mu = 0$, and $T = R$ otherwise, where $\mu \in_r \{0, 1\}$ and $R \in_r \mathbb{G}_1$. The simulator acts as the challenger in the following game:

Initialization: The simulator accepts the DBDH challenge requested by the adversary who selects the $\mathcal{T}^*$.

Setup: The simulator runs the Setup function. It chooses a random $\alpha^* \in_r \mathbb{Z}_p$ and computes the value $\alpha = \alpha^* + ab$. Next, it simulates $e(g, g)^{\alpha} \leftarrow e(g, g)^{\alpha^* + ab} = e(g, g)^{\alpha^*} e(g, g)^{ab}$ and $B = g^{\beta} \leftarrow g^{b}$, where b represents a simulation of the value β. Finally, it sends all components of $PK = \{\mathbb{G}_0, g, B = g^{\beta}, e(g, g)^{\alpha}\}$ to the adversary.

Phase 1: In this phase, the adversary requests multiple private keys $(SK_1, \ldots, SK_{q_1})$ corresponding to q_1 different sets of attributes $(\mathbb{A}_1, \ldots, \mathbb{A}_{q_1})$. After receiving an SK_j query for a given set $\mathbb{A}_j$ where $\mathbb{A}_j \notin \mathcal{T}^*$ (i.e. $\forall A_{j,u} \in \mathbb{A}_j$ does not satisfy $\mathcal{T}^*$), the simulator chooses a random $r'_j \in_r \mathbb{Z}_p$ and defines $r_j = r'_j - b$. Next, it simulates $D_j = g^{(\alpha+r_j)/\beta} \leftarrow g^{\alpha/\beta} g^{r_j/\beta} = g^{(\alpha^*+ab)/b} g^{(r'_j-b)/b}$. Then, $\forall A_{j,u} \in \mathbb{A}_j$, it selects a random $r_{j,u} \in_r \mathbb{Z}_p$ and simulates $D_{j,u} = g^{r_j} \cdot h(u)^{r_{j,u}} \leftarrow g^{r'_j-b} h(u)^{r_{j,u}}$ and $D'_{j,u} \leftarrow g^{r_{j,u}}$. Finally, the simulated values of $SK_j = (D_j, \{D_{j,u}, D'_{j,u} \,|\, r_{j,u} \in_r \mathbb{Z}_p,\ \forall A_{j,u} \in \mathbb{A}_j\})$ are sent to the adversary.

Challenge: The adversary sends two plaintext data files sk_0 and sk_1 to the simulator who randomly chooses a $\mu \in_r \{0,1\}$ by flipping a coin to select one of the files. The simulator then runs the Encrypt function and derives a ciphertext CT^*. It simulates $\tilde{C} = sk_\mu \cdot e(g,g)^{\alpha \cdot sk_\mu} \leftarrow sk_\mu \cdot e(g,g)^{(\alpha^*+ab)c} = sk_\mu \cdot T e(g,g)^{\alpha^* c}$, where c represents a simulation of the value sk_μ and $T = e(g,g)^{abc}$. Next, it simulates $C = g^{\beta \cdot sk_\mu} \leftarrow g^{bc}$. Finally, for each attribute $x \in X^*$ (set of leaf nodes in $\mathcal{T}^*$) it computes $C_x = g^{q_x(0)}$ and $C'_x = h(x)^{q_x(0)}$. The simulated values of $CT^* = \{\mathcal{T}^*, \tilde{C}, C, \forall x \in X^* \colon C_x, C'_x\}$ are then sent to the adversary.

Phase 2: Repeat *Phase 1* with the restriction that the requested private keys are associated with attribute sets such that, $\forall \mathbb{A}_j \mid q_1 + 1 \leq j \leq q$ and $\mathbb{A}_j \notin \mathcal{T}^*$.

Guess: The adversary tries to guess the value μ. If the adversary guesses the correct value, the simulator outputs 0 to indicate that $T = e(g,g)^{abc}$, or 1 to indicate that $T = R$, a random group element in $\mathbb{G}_1$.

Given a simulator $\mathcal{A}$, if $T = e(g,g)^{abc}$, then CT^* is a valid ciphertext, $Adv = \varepsilon$ and

$$Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c, T = e(g,g)^{abc}\right) = 0\right] = \frac{1}{2} + \varepsilon. \tag{2.13}$$

If $T = R$ then $\tilde{C}$ is nothing more than a random value to the adversary. Therefore,

$$Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c, T = R\right) = 0\right] = \frac{1}{2}. \tag{2.14}$$

From equation (2.13) and equation (2.14), we can conclude that

$$\left|Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c, T = e(g,g)^{abc}\right) = 0\right] - Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c, T = R\right) = 0\right]\right| = \varepsilon. \tag{2.15}$$

Therefore, the simulator plays the DBDH game with a non-negligible advantage and the proof is complete. □

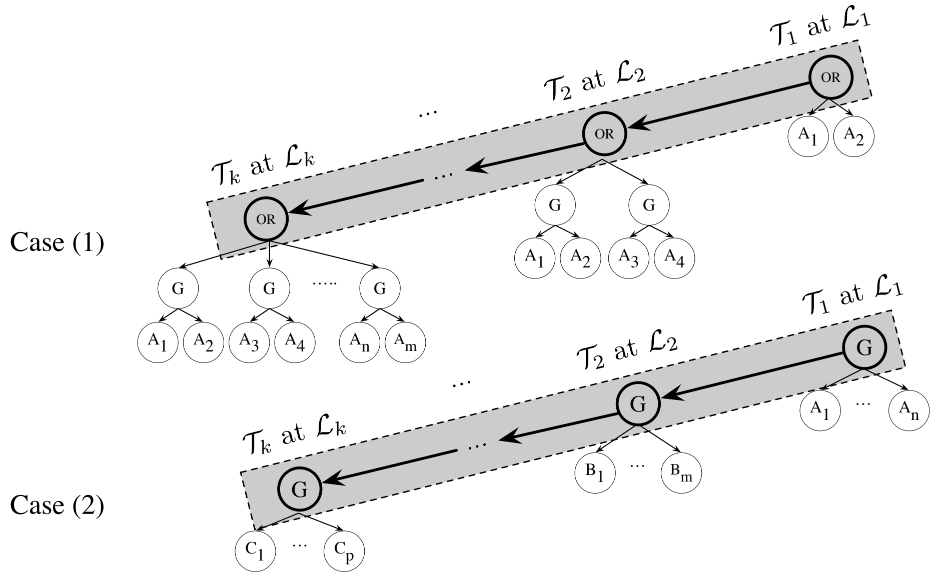
2.5 Performance Analysis

In this section, we present a performance analysis for P-MOD and compare it with three existing schemes, CP-ABE [6], HABE [24] and FH-CP-ABE [31].

2.5.1 Traditional CP-ABE in a Hierarchical Setting

CP-ABE [6] handles the sharing of independent pieces of data based on independent access policies. It was not designed to support a privilege-based access structure (i.e. hierarchical organization). Therefore, to adapt CP-ABE to a privilege-based access structure, the Encrypt function runs once for each level. However, if it were to be used in a hierarchical organization, there would be a trade-off between the key management and the complexity of the encryption and decryption processes. Fig. 2.4 shows the two general cases in which CP-ABE is used to share data with users in a hierarchical organization.

In case (1), key management is favored over encryption and decryption complexities. The



$\mathcal{T}_1$ at $\mathcal{L}_1$
$\mathcal{T}_2$ at $\mathcal{L}_2$
$\mathcal{T}_k$ at $\mathcal{L}_k$
Case (1)
OR
G
A_1
A_2
A_3
A_4
A_n
A_m
Case (2)
B_1
B_m
C_1
C_p

Figure 2.4: CP-ABE used in a hierarchical organization.

individuals at each level possess attributes that define their privileges and the private keys to encrypt the data files. The size of each private key is therefore optimized and the key management process becomes less resource-intensive. However, since levels are independent, attributes must be repeatedly incorporated into each access tree. As a result, the sizes of the access trees at lower-levels will increase, as shown in case (1) of Fig. 2.4. For complex organizations with a large number of levels, the access trees will become even larger. This results in an increase in encryption and decryption complexities.

On the other hand, case (2) favors minimizing encryption and decryption complexities over key management. Each data file is encrypted under an access policy with a set of unique attributes at each level without considering privileges and relationships. This results in simpler access trees at each level of the hierarchy and therefore lower encryption and decryption complexities. However, to grant access to an individual at a specific level, the key-issuer must generate a private key for that individual that incorporates the attributes at that level and all the levels below. Complex hierarchies that include a large number of attributes, could result in complicated key management. As a result, private keys will require incorporating a large number of attributes.

2.5.2 Computational Cost

We formulate the encryption and decryption costs based on the number of group operations $f_{\mathbb{G}_0}$, $f_{\mathbb{G}_1}$ for groups $\mathbb{G}_0$, $\mathbb{G}_1$ respectively and the number of bilinear mapping operations e involved in the Encrypt and the Decrypt functions for each scheme. Table 2.2 summarizes the number of operations for each scheme.

2.5.2.1 Encryption Cost

Encryption cost is measured as the number of basic operations involved in generating the ciphertext from the plaintext. It is formulated based on the Encrypt function that involves group operations $f_{\mathbb{G}_0}$ and $f_{\mathbb{G}_1}$.

The number of operations involved in sharing an independent piece of data using CP-ABE is $(2|X|+1)$ and 2 for $f_{\mathbb{G}_0}$ and $f_{\mathbb{G}_1}$ respectively, where $|X|$ denotes the number of leaf nodes (attributes) of the access tree $\mathcal{T}$. For a hierarchical organization, we present the encryption complexity of case (1) in Fig. 2.4 as it involves a relationship between all levels. In a real-life application, case (2) would not satisfy a hierarchical organization as it requires attributes to be shared by all users regardless of which level they belong to. As shown in Table 2.2, the number of operations involved in the encryption process is formulated as $(2(|X_1| + \cdots + |X_k|) + k)$ and $2k$ for $f_{\mathbb{G}_0}$ and $f_{\mathbb{G}_1}$ respectively, where $|X_1|, |X_2|, \cdots, |X_k|$ are the number of leaf nodes (attributes) associated with access trees $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_k$ respectively. However, the reuse of attributes needed at each level in this scheme increases the computational complexity, making it an overall inefficient solution for hierarchical organizational structures.

Similarly, P-MOD generates a ciphertext for each level of the hierarchy. The number of operations involved is $(2(|Y_1| + \cdots + |Y_k|) + k)$ and $2k$ for $f_{\mathbb{G}_0}$ and $f_{\mathbb{G}_1}$ respectively, where $|Y_1|, |Y_2|, \cdots, |Y_k|$ are the number of leaf nodes (attributes) associated with access trees $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_k$ respectively. However, P-MOD leverages a privilege-based access structure as discussed in Section 2.3.2. This results in smaller sized and level-specific access trees that minimize the number of attributes where, $|Y_i| < |X_i|, \forall i \in k$.

When comparing P-MOD with hierarchical schemes such as HABE and FH-CP-ABE, P-MOD minimizes the overall number of operations. This is because the encryption process for schemes such as HABE and FH-CP-ABE involve more complex hierarchies and access trees that contain all the access policies for all the levels. On the contrary, P-MOD involves smaller access trees, each one limited to level-specific attributes and policies.

In HABE, a single hierarchy represents the root master, domain masters, users, and attributes. This may result in complex hierarchies as the number of users increases. Therefore, the size of $|X|$ may end up being large making the encryption process expensive. Similarly, FH-CP-ABE [31] uses a single access tree $\mathcal{T}$ to encrypt all data files to be shared. The number of operations involved are $(2|X| + k)$ and $(2v|\mathbb{A}_T| + 2k)$ for operations $f_{\mathbb{G}_0}$ and $f_{\mathbb{G}_1}$ respectively, where $|\mathbb{A}_T|$ is the number of transport nodes (levels), and v is the number of children nodes associated with a transport node. This may also result in large sets $|X|$, $|\mathbb{A}_T|$ and v and lead to an expensive encryption process.

2.5.2.2 Decryption Cost

Decryption cost is measured as the number of basic operations involved in decrypting the ciphertext into plaintext. It is formulated based on the Decrypt function that involves bilinear operations e and group operations $f_{\mathbb{G}_1}$.

For a single Decrypt run, CP-ABE [6] involves $(2|\mathbb{A}_j|)$ and $(2|S|+2)$ number of operations e and $f_{\mathbb{G}_1}$ respectively, where $|\mathbb{A}_j|$ is the number of attributes possessed by the j^{th} data user and $|S|$ is the least number of interior nodes that satisfy $\mathcal{T}$. However, to adapt CP-ABE to a privilege-based access structure, the Decrypt function is run as many times as the number of ciphertexts a data user wishes to decrypt. The number of operations involved are $k(2|\mathbb{A}_j|+1)$ and $(2[|S_1|+\cdots+|S_k|]+2k)$ for operations e and $f_{\mathbb{G}_1}$ respectively, where $|S_1|, |S_2|, \cdots, |S_k|$ are the least number of interior nodes that satisfy the access trees $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_k$ respectively.

The number of operations involved in the decryption process for P-MOD is similar to a single CP-ABE decryption run. P-MOD needs to run the Decrypt function only one time, even if the data user needs to obtain more than one data file. The Decrypt function requires $(2|\mathbb{A}_j|)$ and $(2|S|+2)$ for operations e and $f_{\mathbb{G}_1}$ respectively. Once the data user successfully decrypts the ciphertext (obtains the symmetric key at his/her level), the user can derive the remaining lower-level keys as described in equation (2.2). The complexity of the operations involved in deriving the symmetric keys and decrypting ciphertexts at the lower-levels are negligible in comparison with the group and bilinear operations involved in running the Decrypt function, and therefore could be ignored.

When comparing P-MOD to HABE, the e number of operations involved are slightly similar. However, the $f_{\mathbb{G}_1}$ number of operations required are $|\mathbb{A}_j-1|$ and $2|S|+2$ respectively. Therefore, for both schemes, encryption complexity would greatly depend on the number of attributes possessed by the user and the access tree generated during encryption.

FH-CP-ABE [31] aims to satisfy a certain transport node (level) of the single access tree $\mathcal{T}$ allowing the data user to decrypt certain encrypted files up to that level. The number of operations involved in this process are $(2|\mathbb{A}_j|+1)$ and $(2|S|+v|\mathbb{A}_T|+2k)$ for operations e and $f_{\mathbb{G}_1}$ respectively. Again, since FH-CP-ABE uses a single access tree $\mathcal{T}$, the number

Table 2.2: P-MOD comparison of number of operations.

Function	Op.	CP-ABE [6]	HABE [24,25]	FH-CP-ABE [31]	P-MOD
Encrypt	f_{G_0}	$2(\lvert X_1\rvert + \cdots + \lvert X_k\rvert) + k$	$\lvert X\rvert$	$2\lvert X\rvert + k$	$2(\lvert Y_1\rvert + \cdots + \lvert Y_k\rvert) + k$
	f_{G_1}	$2k$	1	$2v\lvert \mathbb{A}_T\rvert + 2k$	2
Decrypt	e	$k(2\lvert \mathbb{A}_j\rvert + 1)$	$3\lvert \mathbb{A}_j\rvert$	$2\lvert \mathbb{A}_j\rvert + 1$	$2\lvert \mathbb{A}_j\rvert$
	f_{G_1}	$2(\lvert S_1\rvert + \cdots + \lvert S_k\rvert) + 2k$	$\lvert \mathbb{A}_j\rvert - 1$	$2\lvert S\rvert + v\lvert \mathbb{A}_T\rvert + 2k$	$2\lvert S\rvert + 2$

Table 2.3: P-MOD comparison of private key and ciphertext sizes.

Comp	Len	CP-ABE [6]	HABE [24,25]	FH-CP-ABE [31]	P-MOD
Priv Key	$L_{\mathbb{G}_0}$	$2\lvert \mathbb{A}_j\rvert + 1$	$\lvert \mathbb{A}_j\rvert$	$2\lvert \mathbb{A}_j\rvert + 1$	$2\lvert \mathbb{A}_j\rvert + 1$
Ciphertext	$L_{\mathbb{G}_0}$	$2(\lvert X_1\rvert + \cdots + \lvert X_k\rvert) + k$	$\lvert X\rvert$	$2\lvert X\rvert + k$	$2(\lvert Y_1\rvert + \cdots + \lvert Y_k\rvert) + k$
	$L_{\mathbb{G}_1}$	k	1	$v\lvert \mathbb{A}_T\rvert + k$	k

of transport nodes $|\mathbb{A}_T|$ can be large, resulting in higher decryption complexity. The least number of interior nodes $|S|$ that satisfy $\mathcal{T}$ can also be large if the construction of the access tree $\mathcal{T}$ is not optimized. Constructing a single access tree that accommodates a large number of attributes is resource-intensive and could become complicated as the access rules become more sophisticated. When comparing the decryption costs of P-MOD and FH-CP-ABE, the decryption cost of P-MOD depends on $|\mathbb{A}_j|$ and $|S|$ while the decryption complexity of FH-CP-ABE depends on $|S|$, v, $|\mathbb{A}_T|$ and k. The size of the sets in FH-CP-ABE will always be greater than the size of the sets in P-MOD due to the different constructions of access trees in each scheme.

2.5.3 Storage Cost

To evaluate the storage efficiency, we formulate the bit-length of the private keys and ciphertexts generated by each scheme. Table 2.3 represents a comparison of the storage costs in bit-length for all schemes. The bit-length of a single element in $\mathbb{G}_0$, $\mathbb{G}_1$ and $\mathbb{Z}_p$ are denoted as $L_{\mathbb{G}_0}$, $L_{\mathbb{G}_1}$ and $L_{\mathbb{Z}_p}$ respectively.

As shown in the table, the bit-length of the private keys for all schemes are similar. In terms of space complexity, this can be reduced to $\mathcal{O}(\mathbb{A}_j)$. On the other hand, the size of the ciphertexts differ. The size of a ciphertext is based on the output of the Encrypt function for each scheme.

For CP-ABE, the total size of all generated ciphertexts from all levels consists of $(2(|X_1| + \cdots + |X_k|) + k)$ elements from $\mathbb{G}_0$ and k elements from $\mathbb{G}_1$. Using this scheme, the lower-levels must accommodate attributes of the higher levels. Ciphertext size can potentially end up large in size due to attribute replication at each level.

For HABE, the ciphertext consists of $|X|$ elements from $\mathbb{G}_0$ and 1 element from $\mathbb{G}_1$. Similar to the discussion in the encryption cost of HABE, the size of X can be large due to the complexity of how the hierarchy is defined. Likewise, the single ciphertext generated by FH-CP-ABE [31] consists of $(2|X| + k)$ elements from $\mathbb{G}_0$ and $(v|\mathbb{A}_T| + k)$ elements from $\mathbb{G}_1$. In this scheme, the ciphertext size depends on $|X|$, $|\mathbb{A}_T|$, v and k. As the size of these sets grows, the ciphertext size can grow exponentially based on how the tree $\mathcal{T}$ is constructed.

P-MOD generates ciphertexts in a similar approach to those generated by CP-ABE. The total size of all generated ciphertexts consists of $(2[|Y_1| + \cdots + |Y_k|] + k)$ elements from $\mathbb{G}_0$ and k elements from $\mathbb{G}_1$. However, the size of the ciphertext generated by P-MOD is shown to be smaller in size than CP-ABE in all instances. This is based on the composition of our proposed access structure that does not duplicate attributes, therefore generates smaller ciphertexts.

2.6 Empirical Results

In this section, the results of various simulations are presented to support the performance analysis discussed in Section 2.5. P-MOD is implemented and simulated in Java using the CP-ABE toolkit [46] and the Java Pairing-Based Cryptography library (JPBC) [47]. For comparison, simulations are also conducted for CP-ABE [6] and FH-CP-ABE [31] under the same conditions as P-MOD. All simulations are conducted on an Intel(R) Core(TM) i5-4200M at 2.50 GHz and 4.00 GB RAM machine running the Windows 10 OS.

In the simulations, the number of levels k is equivalent to the number of file partitions being shared. The total number of different user attributes applied to users across all levels within H is represented as N. Both variables are compared for key generation, encryption, and decryption time-costs. The experiments include applying the values for $k = \{3, 6, 9\}$, to each value of $N = \{10, 100\}$, resulting in 6 experimental cases. Within each case, it is possible to distribute the user attributes among the levels in numerous ways. For example, when testing the schemes with a hierarchy H for $k = 3$ and $N = 10$, there are 36 different ways to distribute the 10 user attributes among the 3 levels, excluding the cases of having zero user attributes at any level. A possible way of distributing them could be $\mathbb{A}_{\mathcal{L}_1} = 3$, $\mathbb{A}_{\mathcal{L}_2} = 3$ and $\mathbb{A}_{\mathcal{L}_3} = 4$, where $\mathbb{A}_{\mathcal{L}_1}$ represents the set of user attributes possessed by users in the highest level within H and $\mathbb{A}_{\mathcal{L}_3}$ the set of user attributes possessed by users in the lowest level. As the values of k and N increase, the possible ways of user attribute distribution among all levels increase. Without loss of generality, we assume that the number of attributes for each level follows the normal distribution in our simulations.

The data set used in the simulations is sampled from the Census Income data set [42]. The sample consists of 30,163 records, and each record is composed of 9 different record attributes. The records are stored in a Microsoft Excel file $\mathcal{F}$, with a total size of 2.42MB. For simulation purposes, it is assumed that data sensitivity is defined over record attributes (vertical columns), corresponding to the second approach described in Section 2.3.1.

$\mathcal{F}$ is partitioned into k sections, where $\forall F_i \in \mathcal{F}$ and $1 \leq i \leq k$, each F_i represents one or more full columns of $\mathcal{F}$. Table 2.4 represents the 9 record attributes distribution of $\mathcal{F}$ into each partition F_i, based on the given value for k in each scenario.

Table 2.4: Attribute distribution in each partition F_i.

k	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9
3	2	3	4	-	-	-	-	-	-
6	1	1	1	2	2	2	-	-	-
9	1	1	1	1	1	1	1	1	1

All simulations are performed in consideration of users at the most sensitive (highest) level within the hierarchy. This approach considers the most complicated applications, where a user needs to gain access to the entire file. Fig. 2.5 summarizes the time expended by the three schemes to generate a private key for the user, encrypt all partitions of $\mathcal{F}$ and decrypt all partitions respectively, in all 6 scenarios. By analyzing the figures, some observations can be derived. The results are summarized in the following subsections.

2.6.1 Key Generation Time-Cost

To measure the time to generate a private key for a user, the same attribute and level conditions are applied to all three schemes. P-MOD outperforms CP-ABE and FH-CP-ABE in all experimental evaluations. As illustrated in Fig. 2.5(a), the time taken to generate a private key for a user at the highest level in CP-ABE and FH-CP-ABE, is independent of the value of k and remains nearly constant when N is kept constant, for both values of $N = \{10, 100\}$ tested.

In comparison, P-MOD reacts differently. When the total number of user attributes is normally distributed among the levels, the time taken to generate a private key by P-MOD is approximately the reciprocal of the value of k multiplied by the equivalent time taken by CP-ABE or FH-CP-ABE to perform the same function. For example, the time taken by CP-ABE and FH-CP-ABE in the case where $k = 6$ and $N = 100$ is approximately 2.2

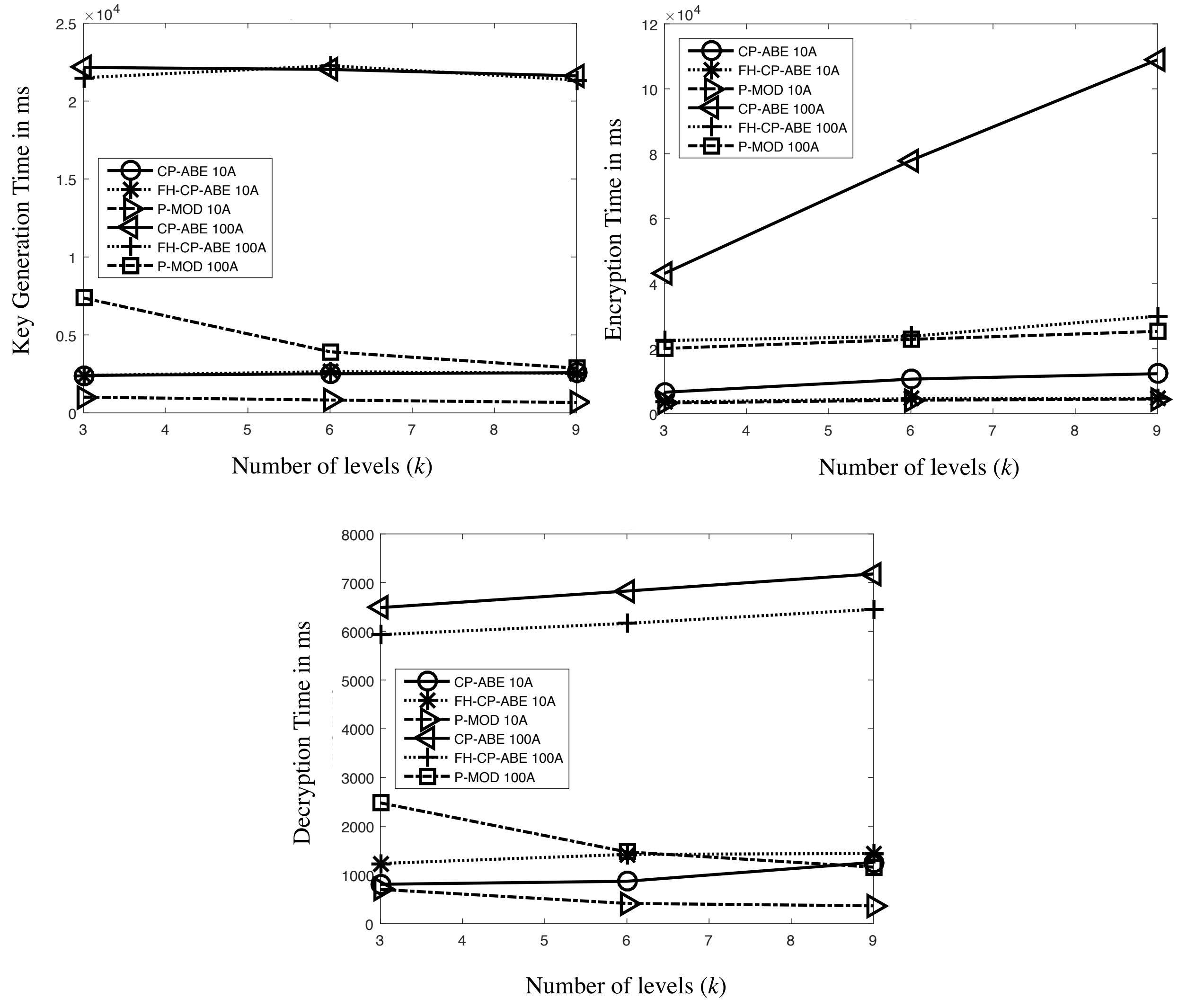


Figure 2.5: P-MOD performance comparison: (a) Key generation time, (b) Encryption time, and (c) Decryption time.

seconds. The time taken by P-MOD in the same conditions is approximately 0.35 seconds, that is nearly $\frac{1}{6}$ times the time-cost of the other schemes. Therefore, the time taken to generate a private key by P-MOD is inversely proportional to the value of k in the hierarchy. This is true for a constant value N with normally distributed attributes.

Another observation that can be made from Fig. 2.5(a) is the effect of the values N on the time expended. The time-cost is directly related to the value N and the degree of this proportional increase is different for each scheme as the value N increases.

We define variable δ as the difference in time-cost of two experimental evaluations of the same scheme at $N = 10$ and $N = 100$, while keeping k fixed. In Fig. 2.5(a), consider the simulated values for each scheme at $k = 9$. Both CP-ABE and FH-CP-ABE result in large time-cost changes, approximately $\delta_{\text{CP-ABE}} = 19039$ms and $\delta_{\text{FH-CP-ABE}} = 18821$ms, while P-MOD results in a smaller value, $\delta_{\text{P-MOD}} = 2204$ms. Based on these values, $\delta_{\text{P-MOD}}$ is approximately 11.7% of $\delta_{\text{CP-ABE}}$ and $\delta_{\text{FH-CP-ABE}}$, resulting in an approximately 88.3% improvement. The significance of this observation can be seen in applications where N is a large value. Efficiency can be gained in time expenditure by using P-MOD, versus CP-ABE and FH-CP-ABE.

2.6.2 Encryption Time-Cost

The encryption time-cost is the time it takes each scheme to perform the encryption function over all partitions of $\mathcal{F}$. Fig. 2.5(b) represents the time expenditure of each scheme under all six experimental scenarios. P-MOD surpasses both CP-ABE and FH-CP-ABE in every experimental case. For example, compare the time duration of the three schemes at $k = 9$ and $N = 100$. The time duration for CP-ABE is approximately 4.3 times of P-MOD to perform encryption. Similarly, FH-CP-ABE is approximately 1.2 times of P-MOD to perform encryption.

All schemes follow a direct proportional pattern as the values k and N increase. These results prove the correctness of the encryption computational complexity analysis presented in Section 2.5.2.1. The encryption function in all schemes involves a number of f_{G_0} and f_{G_1} operations that are dependent on both the values k and N. However, based on the proposed hierarchical access structure, P-MOD is able to outperform both CP-ABE and FH-CP-ABE. In addition to this, the effect of changing the value N is also illustrated clearly in Fig. 2.5(b). For example, when $k = 9, \delta_{\text{P-MOD}}$ is approximately 21.6% of $\delta_{\text{CP-ABE}}$ and approximately 82% of $\delta_{\text{FH-CP-ABE}}$.

2.6.3 Decryption Time-Cost

As previously discussed, the experiments are performed from the perspective of a user that appears at the highest level of the hierarchy. Taking this into account, the decryption time-cost is defined as the time for the user to successfully decrypt all ciphertexts EF_i corresponding to all partitions F_i, if the user possesses the correct set of user attributes. Fig. 2.5(c) illustrates the time to perform the decryption function by each scheme. The decryption function of both CP-ABE and FH-CP-ABE both involve e and f_{G_1} operations that are dependent on the values k and N. As these values increase, the decryption time-cost increases linearly for both schemes, proving the correctness of the decryption complexity analysis in Section 2.5.2.2.

When measuring the decryption time-cost, P-MOD outperforms both CP-ABE and FH-CP-ABE. This is due to the time-cost being inversely proportional to the value of k. The time-cost decreases as the value of k increases while N is kept constant.

The decryption time-cost of P-MOD does not severely increase while the value N changes from 10 to 100. In contrast to this, CP-ABE and FH-CP-ABE are greatly affected, as seen in Fig. 2.5(c). For example, when $k = 9$, $\delta_{\text{P-MOD}}$ is approximately 13.4% of $\delta_{\text{CP-ABE}}$ and approximately 15.8% of $\delta_{\text{FH-CP-ABE}}$. The decryption time-cost of P-MOD is expected to drop when k increases while keeping the file size constant.

In summary, for a hierarchical organization with many levels, the simulation results show that P-MOD is significantly more efficient at generating keys, encryption, and decryption than that of both CP-ABE and FH-CP-ABE schemes.

2.7 Summary

The numerous benefits provided by the cloud have driven many large multilevel organizations to store and share their data on it. In this chapter, we first pointed out major security concerns data owners have when sharing their data on the cloud. To address the concerns, we proposed a Privilege-based Multilevel Organizational Data sharing scheme (P-MOD) that allows data to be shared efficiently and securely on the cloud. P-MOD partitions a data file into multiple segments based on user privileges and data sensitivity. Each segment of the data file is then shared depending on data user privileges. We formally proved that P-MOD is secure against adaptively chosen plaintext attack assuming the DBDH assumption holds. Our comprehensive performance comparison with the two most representative schemes showed that P-MOD can significantly reduce the computational complexity while minimizing the storage space.

Chapter 3

Bitcoin and Blockchain

3.1 Introduction

In Chapter 2, we showed that we can significantly reduce the computational complexity when sharing data in complex hierarchical organizations. However, our P-MOD scheme relies on a cloud storage system which represents a single point of failure. Therefore, our next goal is to eliminate reliance on a central system. In order to achieve that, we expand our scheme into a blockchain-based system.

Before we delve into the details of this scheme, in this chapter, we review and analyze the major security issues of Bitcoin and its underlying foundation, blockchain. We begin by presenting a comprehensive background of Bitcoin and the preliminaries on security. Next, we investigate double-spending attacks that pose a major security threat to Bitcoin. The main purpose of this research is twofold. We aim at analyzing the security of blockchain to understand the potential of developing schemes that build over it. We also provide an analysis that presents a trade-off between the waiting time before accepting a transaction in Bitcoin versus the profits/losses of the attackers. This analysis can act as an indicator to determine how secure schemes that build atop blockchain may be.

The rest of this chapter is structured as follows. In Section 3.2, we provide a comprehensive background review on Bitcoin. Next, in Section 3.3, we evaluate double-spending attacks and present our profitability analysis. Finally, in Section 3.4, we conclude our study.

3.2 Understanding Bitcoin

Blockchain as a concept was initially proposed by Stuart Haber and W. Scott Stornetta in 1991 [48]. Their blockchain aimed at certifying the creation/modification of a digital record by digitally time-stamping the records. However, the blockchain was not efficient since each record was independently time-stamped. To improve efficiency, Merkle trees [49] were incorporated into blockchains in 1992 [50]. They improved efficiency by handling multiple digital records into one block. Finally, Satoshi Nakamoto implemented the first real blockchain and used it as the core technology for the Bitcoin cryptocurrency system [51]. In this section, we will present the major building blocks and protocols of Bitcoin.

3.2.1 The Bitcoin Network

Bitcoin runs over a P2P network. The main advantage of using a P2P network is the agile movement of data for all nodes to achieve consensus. In contrast to the typical P2P network used to share data files between interested peers, Bitcoin uses the network to rapidly broadcast data among all the connected nodes. This process is known as *flooding* and continues until all nodes within the network receive the broadcast data.

It is important to differentiate between the terms *node* and *peer* of a P2P network. A node is a network entity that is connected to one or multiple other similar nodes. The directly connected nodes are referred to as peers. Nodes propagate data to indirectly connected nodes by forwarding it to their peers until the data reaches every connected node. In the Bitcoin network, data being flooded includes IP addresses of the nodes, newly generated transactions, and blocks of verified transactions that extend the blockchain. Peers share IP addresses of other nodes that they are connected to or have discovered from their peer nodes. The goal behind sharing IP addresses is to allow peers in the network to discover and connect to more nodes resulting in a random network topology. Newly generated transactions are broadcast through the network to rapidly publicize their occurrence to all connected nodes. Miners compete to mine these transactions into blocks. The winning miner broadcasts the block to all the connected nodes to extend and update their version of the blockchain.

Nodes in the Bitcoin P2P network are defined based on their roles. The main duties are summarized as transaction generation, block/transaction routing, block/transaction verification, and transaction mining. Block/transaction routing is performed by all nodes. A node that can perform all functions is referred to as a *full node.* It consistently keeps a copy of the full blockchain allowing it to verify any transaction without needing the assistance of other connected nodes. It also possesses a BTC wallet that can generate transactions and compute the possessed value of BTC by the node. Moreover, a full node possesses computational resources to compete in the mining competition. Nodes that do not store a full copy of the blockchain are referred to as *Simplified Payment Verification (SPV) nodes* or *lightweight nodes* [52]. These nodes require assistance from full nodes when verifying a transaction. Full nodes feed the SPV nodes with the required information from the blockchain necessary to complete the transaction verification. Some nodes may only perform one particular function. Ones that are engaged in the mining process are referred to as *mining nodes* while others that generate transactions are referred to as *wallets.*

In most Bitcoin software implementations, all nodes are treated equally and are uniquely identified by their IP addresses. Using these addresses, peers establish Transmission Control Protocol (TCP) connections with one another. Each node can choose whether to connect to the network using a public or private IP. A node that uses a public IP is accessible over the Internet by any node while one with a private IP is only accessible by nodes within its private network. By default, a node with a public IP address is granted 8 *outbound* connections and 117 *inbound* connections, resulting in a total of 125 connections. On the other hand, a node with a private IP address is granted only 8 *outbound* connections. An outbound connection is initiated by the node itself when it requests connecting to a discoverable node while an inbound connection is initiated by other nodes in the network that desire connecting to the node.

The node that initiates a connection is defined as *client* and the node that waits for an incoming connection as *server*. Both nodes engage in a TCP handshake by exchanging network packets defined as `version` and `verack`. The client initiates a connection request by sending a `version` packet addressed to the IP address of the server. By default, the server listens on port 8333 for incoming `version` packets. If the server accepts the `version` packet, it responds with a `verack` packet and its own `version` packet, both addressed to the IP address of the client. Finally, the client responds by sending a `verack` packet addressed to the IP address of the server and the connection is established. The connection enables symmetric communication allowing the client and server to exchange data bidirectionally. The connection is lost if peers do not communicate for a specified idle time. To reconnect, peers engage in a new TCP handshake.

As discussed previously, a node shares with its peers a list of IP addresses that it has learned as a result of being connected to the network. Each node stores its list in two separate tables: a *tried* table and a *new* table. The tried table of a node stores IP addresses that the node has established connections with while its new table stores IP addresses that it has only discovered but did not attempt to connect to yet. When a node desires sharing IP addresses with its peers, it randomly selects IP addresses from both tables and sends them in addr messages. An addr message can contain up to 23% or a maximum of 1000 IP addresses of the total IP addresses stored in both tables [53]. To initiate sharing, a node sends a getaddr message to its peers requesting them to share their lists of IP addresses. The peers then respond with an addr message. In some cases, sharing IP addresses is unsolicited if a node voluntarily sends an addr messages to its peers without receiving a getaddr message.

A node that wishes to connect to the Bitcoin network for the first time cannot obtain IP addresses by this method. Bootstrapping is mainly achieved by communicating with a Domain Name Server (DNS) seeder. The node sends a DNS query requesting a list of active IP addresses. If the DNS fails to respond with an appropriate list of active IP addresses, the node can still connect to the network by using a hard-coded list of IP addresses, referred to as *seeds*. Once connected to any of these IP addresses, the node can then request more IP addresses from its peers by sending getaddr messages.

Nodes also relay verified transactions and blocks to their peers to reach consensus. A node begins by broadcasting an inventory (inv) message to all its peers informing them of the new transactions/blocks it has received and verified. The peers check whether they are already informed of these new transactions and blocks. Then, they respond to the node with a getdata message. The getdata message includes all the transactions and blocks that a peer node is not aware of. The node then responds with a transaction/block message that includes the complete transactions/blocks the peer requests. Once received, the peer validates the transactions or blocks and continues to relay them to its peers in a similar manner. If a received transaction or block cannot be validated, it is immediately dropped and its propagation is discontinued.

3.2.2 Bitcoin Transactions

We define a Bitcoin transaction (TX) as the transfer of an amount of BTC *ownership rights* from the wallet of the buyer to the wallet of the seller, in exchange for a product or service. BTC wallets use elliptic curve digital signatures to handle the transfer of ownership rights and ensure that unauthorized spending of the cryptocurrency is infeasible. Each wallet randomly generates a private key Pr, that is used to derive its corresponding public key Pub that is shared among all users. The Pub is used to generate the address of the wallet needed to make payments to it while the Pr is used to generate a digital signature corresponding to the Pub in order to claim payments made to the wallet and use them in later transactions. A Pr is first generated from a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) and its corresponding Pub is then calculated using Elliptic Curve Digital Signature Algorithm (ECDSA). Calculations are performed based on the field and curve parameters defined by $\mathsf{secp256k1}$ with the curve order n [54] as follows

$$\mathsf{Pr} = \mathsf{CSPRNG}(), \tag{3.1}$$

$$\mathsf{Pub} = \mathsf{Pr} \times G \pmod{n}, \tag{3.2}$$

where G is a generator of the elliptic curve and $\times$ represents elliptic curve multiplication.

The BTC wallet of the buyer assembles a transaction using the Unspent Transaction Outputs (UTXO) of the buyer stored in the blockchain. An UTXO specifies an amount of BTC claimed earlier by the buyer as a result of a previously processed transaction. A simple BTC transaction is shown in Fig. 3.1. In the figure, we show that a transaction can consist of multiple inputs and outputs. The output UTXO_{pay} represents the transfer of ownership rights of a certain amount of BTC from the wallet of the buyer to the wallet of the seller.

The output UTXO_{ch} represents redirecting ownership rights of the BTC change amount back to the wallet of the buyer. A distinct *locking script* is attached to each of these outputs that specifies conditions that must be met in order to grant ownership rights. For example, the locking script attached to UTXO_{pay} must include the Pub of the seller needed to generate his/her wallet address. This ensures that the payment is made to the wallet of the seller and only he/she is granted access to it with his/her corresponding Pr. Using Pr, the seller can generate a digital signature that corresponds to the Pub associated with the locking script, hence claim the output.

The inputs $\{\mathsf{UTXO}_1, \mathsf{UTXO}_2, \cdots, \mathsf{UTXO}_n\}$ represent unspent transaction outputs claimed by the buyer from previous transactions. When a buyer decides to use a specific output from a previous transaction as an input to a new transaction, the buyer must specify proof that he/she still possesses ownership rights and did not previously spend them in another transaction. This is done by attaching an *unlocking script* to each input. The unlocking script solves the locking script that was associated with the output from the previous transaction. Likewise, the unlocking script is a digital signature produced by the Pr of the buyer that corresponds to a Pub associated with the locking script of an UTXO. A valid unlocking script is legitimate proof of continuous possession of ownership rights to certain BTC being used as input. As a result, BTC can be viewed as a chain of digitally signed transactions where ownership rights are transferred from one owner to the other by digitally signing them.

A transaction must include at least one input, however, it may include multiple outputs to simultaneously pay different sellers from the total value associated with the inputs. The locking script of each output would specify the conditions of its claimer. However, it is necessary that the total BTC value of the inputs is always equal to or greater than the total value of the outputs. In the event that the total value of the inputs is greater than the total

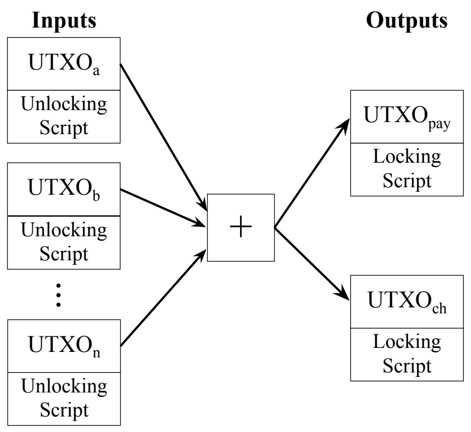


Figure 3.1: A single Bitcoin transaction with multiple UTXO inputs and outputs.

outputs, the difference, known as the *transaction fee*, is rewarded to the miner that adds the transaction into a block attached to the blockchain. For guaranteed processing, most available wallets today derive the transaction fee as a fixed amount of BTC in relation to the size of the transaction. In other words, the transaction fee increases with the size of the transaction.

The wallet of the user combines all the transaction inputs/outputs and their corresponding scripts into one digital message M. It then applies the Secure Hash Algorithm $\mathsf{SHA256}$ to M twice to increase security before releasing it into the network. The 32 byte digest representing the identity of the transaction ($\mathsf{ID}_{\mathsf{TX}}$) is generated as follows

$$\mathsf{ID}_{\mathsf{TX}} = \mathsf{SHA256}\left(\mathsf{SHA256}(\mathsf{M})\right). \tag{3.3}$$

A newly generated transaction assembled by the BTC wallet of a buyer is released into the Bitcoin network to be validated and stored in the blockchain. The generating node transfers the transaction to its peers that flood it to the rest of the network nodes. Each node that receives it audits the inputs by executing the scripts associated with it. This audit involves checking whether the execution of the unlocking script integrated by a buyer within each input matches its corresponding locking script defined in the previous transaction. If a match exists, the node relays the transaction to its peers and temporarily places it in its *transaction pool* until chosen to be mined, otherwise, the transaction is dropped.

In some cases, transactions are not flooded into the network in the same order they are generated. As a result, during the audit, a node might not be aware of some inputs of a transaction (child transaction) referring to the outputs of other transactions (parent transactions). Instead of immediately rejecting the transaction and considering its inputs as invalid, the node can temporarily place it into an *orphan transaction pool.* If the parent transaction shows up, the inputs of the child transaction become valid and it can be transferred to the transaction pool.

3.2.3 Bitcoin Transaction Standards

Currently, there are five Bitcoin transaction standards and a few non-standard transactions [52]. All transaction types are generated with a stack-based scripting language that is processed from left to right. A script consists of a list of instructions that must be executed in the correct order to grant an individual the right to spend the BTC within a transaction. The list of standards is described below.

Pay to Public Key Hash (P2PKH): This standard transaction is the most used type. The locking script within each output of a transaction holds the public key hash (serving as a Bitcoin address) of the seller that will claim the BTC amount included. In other words, the locking script defines a condition that the seller must possess a specific Pr corresponding to the public key hash to claim the output. Once claimed by the seller, the output becomes an UTXO owned by the seller. In order for the seller to use this specific UTXO as an input to a future transaction, the seller must attach a valid unlocking script to it. The unlocking script includes the Pub of the seller and a digital signature generated by his/her Pr that corresponds to the public key hash associated with the locking script of the previous transaction output.

Pay to Public Key: The intent behind this standard transaction is to simplify the P2PKH standard. Rather than associating the public key hash within the locking script of the output, the public key itself is used. As a result, the validation process is simple. The digital signature of the seller generated with a Pr can immediately be compared to the associated Pub by searching whether or not they match.

Multi-signature (MultiSig): In this standard transaction, a combination of keys is required to authorize an output claim. The locking script of a transaction output is associated with a number (N) of public keys. In order for an individual to claim the output, the individual must possess M-of-N private keys that correspond to the N public keys. This type of transaction can increase security and can be used in scenarios that require more than one user to be present in order to claim and spend BTC. However, as the number N of public keys associated with the transaction output increases, the size of the transaction also increases. As a result, these transactions acquire large space in the UTXO pool, therefore requiring more storage memory. As discussed previously, larger transactions also require larger transaction fees.

Pay to Script Hash (P2SH): This standard transaction was introduced to resolve the complex issues caused by MultiSig transactions. The transaction has the same simple complexity as a P2PKH transaction. Rather than associating the entire locking script with a transaction output that includes multiple public keys, a double hash computation is applied to the entire script, specifically $\mathsf{SHA256}\big(\mathsf{RIPEMD160}(\mathit{script})\big)$. The result is a 20-byte digest that is attached to the locking script instead of the entire original script. In order to use the output from this transaction as an input to another transaction, the buyer creates an unlocking script that holds M-of-N private keys and the original script that was cryptographically hashed earlier. In that way, sufficient information is available in the locking and unlocking scripts to validate the UTXO for spending. In addition, the buyer no longer has to worry about generating large transactions that might require hefty transaction fees to process. Instead, only the seller is required to provide the unlocking script he/she wishes to spend the output in a new transaction.

Data Output: This standard transaction is intended to store arbitrary data on the blockchain rather than transfer BTC from a buyer to a seller. In the Bitcoin community, many members believe that such transactions are abusive to the system since it places a burden on the network nodes to process transactions that do not carry BTC. However, such transactions exist and allow 40 bytes of data to be stored per transaction. These transactions are un-spendable, therefore are not stored in the UTXO set.

Non-Standard: A very small percentage of transactions are processed under non-standard transactions. Non-standard transactions use more sophisticated scripts that strive to provide higher complexity and security. In some cases, these transactions might even be the result of bugs or mistakes resulting in loss of BTC.

3.2.4 Merkle Trees

Validated transactions are grouped into blocks that are then mined and stored in the blockchain. A single block can contain multiple transactions up to the block size limit. Merkle trees, sometimes referred to as hash trees, are used to cluster multiple transactions in one block.

A Merkle tree is a tree data structure generated in a bottom-up approach that can efficiently summarize and verify the integrity of the transactions being combined. Starting from the leaf nodes that are hashes of the original data, each non-leaf node is generated as a computation of its respective children nodes. For a single non-leaf node, all its children nodes are concatenated then hashed to produce a single digest that represents the node in the tree. This approach continues until a single node is generated that is defined as the root node.

BTC uses a binary Merkle tree in which each non-leaf node has exactly two children. It applies a double hash computation $\mathsf{SHA256}\left(\mathsf{SHA256}(\cdot)\right)$ when generating nodes. The leaf nodes used to construct the tree are the identities $\mathsf{ID}_{\mathsf{TX}}$ generated for each transaction as discussed in equation (3.3).

In a binary Merkle tree, each row within the tree consists of an even number of nodes, except the root node. In the case where a row consists of an odd number of nodes, a replica of the last node is reproduced to even out the number of nodes in that row. To better comprehend the construction of the binary Merkle tree, consider a block that consists of five transactions, $\{\mathsf{TX}_1, \mathsf{TX}_2, \cdots, \mathsf{TX}_5\}$. Each one of these transactions has already been validated by the nodes and an identity for each transaction has been generated as discussed in equation (3.3). We denote the corresponding identities as $\{\mathsf{ID}_{\mathsf{TX}_1}, \mathsf{ID}_{\mathsf{TX}_2}, \cdots, \mathsf{ID}_{\mathsf{TX}_5}\}$, where each identity represents a leaf node in the tree. In this example, the number of nodes at the leaf node level is odd, therefore a replica of the fifth identity is generated, $\{\mathsf{ID}_{\mathsf{TX}_1}, \mathsf{ID}_{\mathsf{TX}_2}, \cdots, \mathsf{ID}_{\mathsf{TX}_5}, \mathsf{ID}_{\mathsf{TX}_6} = \mathsf{ID}_{\mathsf{TX}_5}\}$. Next, the double hash computation is applied to the concatenation of each two identities to generate the parent non-leaf nodes of the Merkle tree as follows

$$N_1 = \mathsf{SHA256}\big(\mathsf{SHA256}(\mathsf{ID}_{\mathsf{TX}_1} \| \mathsf{ID}_{\mathsf{TX}_2})\big), \tag{3.4}$$

$$N_2 = \mathsf{SHA256}\big(\mathsf{SHA256}(\mathsf{ID}_{\mathsf{TX}_3} \| \mathsf{ID}_{\mathsf{TX}_4})\big), \tag{3.5}$$

$$N_3 = \mathsf{SHA256}\big(\mathsf{SHA256}(\mathsf{ID}_{\mathsf{TX}_5} \| \mathsf{ID}_{\mathsf{TX}_6})\big), \tag{3.6}$$

where $\|$ is the concatenation of two identities.

As shown in the previous equations, an odd number of non-leaf nodes is generated at that level. To even it out, we replicate N_3 to produce N_4 as

$$N_4 = \mathsf{SHA256}\big(\mathsf{SHA256}(\mathsf{ID}_{\mathsf{TX}_5} \| \mathsf{ID}_{\mathsf{TX}_6})\big). \tag{3.7}$$

Using the resulting digests we can generate the following level of non-leaf nodes as

$$N_5 = \mathsf{SHA256}\big(\mathsf{SHA256}(N_1 \| N_2)\big), \tag{3.8}$$

$$N_6 = \mathsf{SHA256}\big(\mathsf{SHA256}(N_3 \| N_4)\big). \tag{3.9}$$

Finally, the 32 bytes root node is derived as

$$R = \mathsf{SHA256}\big(\mathsf{SHA256}(N_5 \| N_6)\big). \tag{3.10}$$

Fig. 3.2 represents the complete construction of the Merkle tree for this example. The dotted nodes represent the replicated nodes that are added to even out the odd rows. The root node, R, representing the summary of all transactions is placed into the block header of a block to be mined and chained to the blockchain.

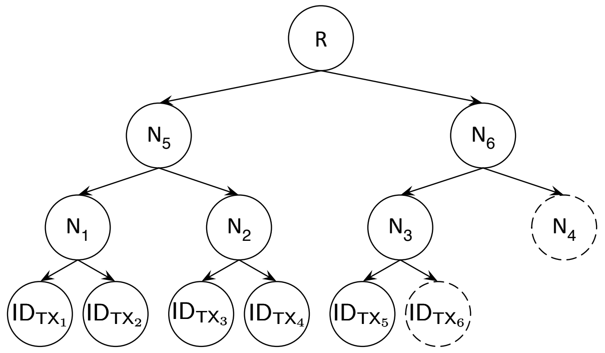


Figure 3.2: A Merkle tree within a block.

The use of Merkle trees is more common in SPV nodes since they do not store a copy of the full blockchain. When an SPV node needs proof of the existence of a transaction within a block, it turns to a full node for assistance. The full node will generate a *merkle path* by computing a maximum of $\log_2 N$ $\mathsf{SHA256}\big(\mathsf{SHA256}(\cdot)\big)$ computations, where N represents the total number of transactions in the tree. Using the Merkle path as an authentication path, the SPV node can prove the existence of a transaction within the tree. This proof of existence method is considered to be efficient since it only requires hash computations.

3.2.5 Blockchain

The blockchain is a public ledger that stores all previous transactions since the creation of Bitcoin. It provides its users with transaction confirmations to track ownership rights of BTC. As new transactions are processed, the blockchain is extended. It consists of blocks $\{\mathsf{B}_0, \mathsf{B}_1, \cdots, \mathsf{B}_n\}$, each carrying a set of validated transactions, where B_0 represents the first block and B_n represents the most recent block attached to the blockchain. Blocks are linked back-to-back, with each one referencing its previous block to form the complete blockchain. To reference a block, a unique 32 byte identity $\mathsf{ID}_{\mathsf{B}_i}$ is generated for B_i by applying $\mathsf{SHA256}\big(\mathsf{SHA256}(\cdot)\big)$ to the block header. An identity is referred to as the block hash.

The head of the blockchain is denoted as B_0 and is defined as the *genesis block*. B_0 differs from all the other blocks as it does not reference any previous block. At the launching stage of the system, B_0 was a stand-alone block waiting for the system to initiate a newly mined block to be chained to it.

Each block consists of two parts, a header, and a body. Each header incorporates the block hash of its predecessor block in the chain. The header also consists of a difficulty target, nonce, and a time-stamp that are discussed in more detail in the following subsection. The body carries all the leaf nodes and non-leaf nodes of the Merkle tree, excluding the Merkle root, which is incorporated in the header. This design makes it infeasible to retroactively alter records within any block of the blockchain. Any modification to one block will require adjusting all the subsequent chained blocks.

3.2.6 Bitcoin Mining

Bitcoin mining is the final stage to secure validated transactions and add them to the blockchain. Once a transaction is added to the blockchain, it becomes completely verified and public to all users. The transaction claimer(s) can use the embedded UTXO(s) as the input to other transactions whenever desired.

Miners begin by selecting transactions from their transaction pools that will be placed into a block where a block cannot exceed 1MB in size. A small portion of that space is specified to carry high priority transactions. Priority is based on the size and age of the transaction inputs. The rest of the block is filled with other transactions that have greater transaction fees to maximize the profit that a miner can turn if successful in mining the block first. A transaction with low or no fees will probably remain in the transaction pool of the miner until it ages and becomes a high priority transaction.

Next, the miner assembles a special transaction, known as the *coinbase transaction*. This transaction is a reward paying transaction to the miner in the event of winning a mining competition. It does not have any inputs and consists of a single output addressed to the wallet of the miner. The amount incorporated in the output is the reward mining fee (12.5 BTC at the time of writing) plus the sum of all transaction fees included in each transaction.

All the selected transactions along with the coinbase transaction are then combined into a Merkle tree as discussed previously. At this point, the miner has all the components needed to construct the block header of the new block except the *nonce*. The nonce is a value that if concatenated with the block header of the group of chosen transactions and then double hashed, it produces a digest with a specific prefix of zeros in its binary representation. Searching for this value is performed in a brute-force manner and is directly correlated with the computational power available. The more available computational power, the faster a miner is able to find the correct nonce. A successful miner will then broadcast his/her *proof-of-work* to prove that he/she consumed computational resources in order to find the correct nonce.

The primary advantage of the proof-of-work is to make it computationally infeasible to perform Sybil attacks [55]. This process is intentionally designed to be resource-intensive to perform while efficient to verify that the work has been done. It is required that a certain number of zeros appear in the prefix of the digest as a result of applying the double SHA256 computation. The prefix determines the *difficulty* of finding the correct nonce. The more zeros required in the prefix of the digest, the harder it is to find the correct nonce and vice versa. The difficulty is dynamically altered every two weeks so that the average time it takes a miner to find a correct solution is approximately ten minutes. As the number of miners increases, the difficulty increases, and vice versa.

The first miner to find the correct nonce to a block of transactions is rewarded a mining reward as compensation for the computational power spent. The mining reward is halved precisely every 210,000 blocks that are added to the blockchain. It is estimated to continue until the year 2140 when nearly 21 million BTC will have been released into the system. The reason for having a fixed supply of BTC is to prevent price inflation in the future.

Another incentive that encourages miners to spend their computational power to perform mining is the transaction fee. The winner is not only rewarded the mining reward but is also given all the transaction fees incorporated with all the transactions in the block. With time, the mining reward will decrease due to halving, which will demand higher transaction fees in the future to compensate for the reduced mining reward.

After a block is successfully mined, all the miners check their transaction pools to eliminate the transactions that have been included in the mined block and immediately construct a new block of transactions. The end of the mining race marks the beginning of a new one. Miners instantly begin to search for the nonce of the next block of transactions.

Simultaneously, the mined block is flooded through the network so that all the nodes can update their blockchains. The winning miner transmits the block to its peer nodes to validate it before propagating it further through the network. The peer nodes check whether the block is correctly assembled in terms of syntax and variables. The proof-of-work provided by the miner must be correct and the coinbase transaction must include the correct amount to pay the miner. If any information is invalid, the block is immediately dropped.

Quite regularly, as blocks are mined to extend the blockchain, a temporary incident, known as a *fork*, might occur. A fork occurs when two miners are able to simultaneously mine two different blocks at the same time. As a result, both newly mined blocks are accepted to extend the blockchain. The blocks are flooded into the network and the miners update their version of the blockchain-based on the block they receive first. This results in two valid versions of the blockchain in possession by the miners with two different paths. However, the miners continue to extend their version of the blockchain regardless of which path they possess. Eventually, one path will grow longer than the other as mining continues. The path that grows longer is the winner and all nodes immediately discard the other path and update their blockchain to the longer one. In literature, the blocks that are dropped are known as *orphan blocks*; valid blocks that were part of the blockchain at some point.

3.2.7 Bitcoin Mining Pools and Payment Methods

Although solo miners can compete in the mining process, the likelihood of a successful return is very low. This is even the case for solo miners with the most powerful computing machines. As a solution to this problem, solo miners collaborate in the mining process by joining computational power into mining pools. Together, they form a large organization with significant computational power that can compete with the other large entities. The members of the mining pool work together to find the correct nonce for a candidate block and report the result as one miner, increasing their chances of winning the competition. In the event of success, the rewards are split among the participating miners based on the contribution provided by each.

The concept of a mining pool can be compared to the lottery. Assuming individuals with the same financial capabilities, if a large group buys tickets together, the individuals within the group have a better chance of winning than a single individual buying tickets alone. If any ticket owned by the group wins the lottery, the participating individuals split the reward proportional to the amount invested by each. In a mining pool, the computational power provided by each solo miner is analogous to the amount invested by each ticket buyer.

A mining pool is managed by a pool operator who handles the entire pool server and receives a percentage of the rewards as compensation. The role of the operator is to coordinate the mining performed by all the participating miners. The operator keeps a continuously updated copy of the entire blockchain to ease the job of the participating miners. Using the updated blockchain, the operator verifies any transaction that appears in the network and places it in a candidate block for mining. By that, miners only need to worry about finding the correct nonce of the candidate block. If the mining pool wins the competition, the operator divides the rewards among the participating miners.

Reward splitting can be performed in multiple forms and varies from one mining pool to the other. As described in [56], these methods can be categorized into simple reward, score-based reward, or risk-free pay-per share reward.

Simple reward systems consist of either proportional systems or Pay-Per Share (PPS) systems. In the proportional systems, a reward B is split among the participating miners at the end of each round, where a round is the consecutive time between two successful blocks generated by the pool. The operator keeps a percentage of the reward fB and divides the remaining $(1-f)B$ among the miners based on the shares they submit. Shares are defined as the number of hashes performed by each miner in attempt to find the correct proof-of-work. A miner that submits n shares from a total of N shares submitted by all the miners in the pool receives a reward of $\frac{n}{N}(1-f)B$ BTC on average. Conversely, the PPS system is a deterministic one where the miner knows how much reward can be turned in advance. The operator immediately pays each miner based on the submitted shares regardless of the mining result. In other words, a miner that submits n shares receives $(1 - f)pB$ BTC/share, where p represents the probability of one share being the correct proof-of-work. In this system, the operator is taking the risk of mining independently since the miners receive ensured payments whether or not the pool generates a block.

Score-based reward systems come in many forms and strive to prevent miners from pool-hopping. Pool-hopping is the practice of mining in a pool only during its good times (successfully generating blocks) and leaving it during its bad times. A pool-hopper can maximize his/her rewards at the expense of miners that remain loyal to the pool at all times. The method introduced by Slush [57] is one of the first implemented score-based systems that extends the proportional method. Rather than paying the miner an amount based on the submitted shares after each round, the miner is given a score that is proportional to his/her contribution and increases as more time elapses from the start of the round. The score is used to calculate the reward share given to the miner at the end of the round. However, this method is still susceptible to hopping since the score does not consider factors such as the mining difficulty or the hashrate of the pool. Also, in this method mining at the beginning of a round is more profitable since there are fewer shares at that time. As a result, the geometric method was introduced to address these weaknesses. This method introduced a fixed fee, a constant amount taken from the reward of each block, and a variable fee, a score granted at the beginning of each round to the operator. As time passes, the variable fee declines, making mining equally profitable throughout the entire round. Shorter rounds result in larger variable fees and vice versa. By that, there is no advantage to mining early in the round.

Another score-based method is Pay-Per-Last-N-Shares (PPLNS) that exists in different forms. In this method, the concept of rewarding miners after each round is replaced with rewarding miners that have been participating in earlier rounds, regardless of the mining result. In other words, the operator pays miners based on their contributions from previous efforts. Later on, more advanced payment systems evolved such as the Double Geometric Method (DGM). This system is a hybrid between the PPLNS and geometric system that combines the advantages of both methods.

Some mining pools employ a risk-free pay-per share system. One of the first implemented systems is known as the Maximum Pay-Per Share (MPPS). It combines both the PPS and proportional systems, where each participating miner has a balance of each. If the miner submits a share, the PPS balance is incremented and when the pool successfully generates a block, the proportional balance is incremented. At pay time, the miner receives the minimum of both balances. This method protects the pool from taking the risk alone. However, this method is inconsiderate to the miners, since they will always make less whether the pool is successful or not. In addition to this, the system suffers from pool-hopping. A solution was later proposed to solve this problem in the Shared Maximum Pay-Per-Share (SMPPS) system. The miners have a PPS balance that continues to accumulate as the miners are participating. If a block is found by the pool and there are sufficient funds, the miners are paid based on their PPS balance. However, if there are no sufficient funds, miners are paid proportional to the available funds and given credit to be paid later for whatever balance that is owed.

Today, a broad range of mining pools exist that give miners a variety of options when joining pools. The question most miners would ask is which mining pool is the best to join. The answer here lies in the preferences of the miners. For example, some miners are not willing to take the risk of not getting paid in the event of being unsuccessful in generating a block and would prefer a PPS mining pool. Others might be willing to take the risk and choose a score-based system for instance, in return for a larger profit.

3.2.8 Alternative Cryptocurrencies

In literature, alternative cryptocurrencies are known as *altcoins*, most of which are inspired by Bitcoin. Altcoins strive to offer innovative features and enhanced security/privacy countermeasures in an effort to compete with Bitcoin. Their development process is based on the level of innovation and security/privacy countermeasures they present.

The simplest method to develop an altcoin is by forking the open-source code of Bitcoin [58] while adding/modifying any features to it. In software development, a *fork* is a completely independent project that exploits a copy of the original source code. A Bitcoin fork generates an entirely new blockchain and is completely independent of Bitcoin. Namecoin [59] is the first developed Bitcoin fork that adopted all of the characteristics of Bitcoin. It also introduced an additional feature allowing users to store data within its transactions. Various Bitcoin forks have evolved latterly with more features and handled security/privacy issues. Many of these forks implemented privacy protocols to increase the anonymity of cryptocurrencies.

On exceptional occasions, an altcoin can also be the result of a *hardfork*. A hardfork occurs when modifications are made to the original software of Bitcoin making its new transactions/blocks incompatible with those previously generated prior to the modifications. These modifications can be as simple as altering certain parameters, such as the block size, or as complex as changing major protocols, such as the consensus algorithm. In order to enforce these modifications, the majority of users/miners must upgrade their client nodes to the latest version that accommodates these changes. The users/miners that do not accept the upgrade will view the new transactions/blocks as invalid and will not accept them. As a result, the blockchain will inevitably split into two paths, one storing transactions of the original cryptocurrency and one storing transactions generated due to the modifications made, hence creating a new altcoin. Users in possession of the original cryptocurrency will automatically be granted an equivalent amount of the new altcoin to what they hold.

Bitcoin Cash [60] is a notable example of a Bitcoin hardfork that occurred on August 1, 2017. It was the result of enforcing BIP91 [61] that proposed activating Segregated Witness (SegWit) [62]. SegWit increases the transaction speed of Bitcoin by splitting the transaction into segments and removing the unlocking signatures that are attached separately at the end. The majority of the miners accepted this proposal resulting in Bitcoin Cash. Users who possessed BTC were immediately granted an equivalent amount of BCC (The currency of Bitcoin Cash) to the BTC they possessed.

While only borrowing the concept of storing transactions in a blockchain, some altcoins have been implemented from scratch with a completely different design and purpose. These altcoins strive to provide services and security/privacy countermeasures beyond the capabilities of Bitcoin or any of its forks. They present substantial differences such as integrating enhanced consensus algorithms or using private (permissioned) blockchains. In contrast to the public (permissionless) blockchain of Bitcoin, where all participating nodes are allowed to execute the consensus protocol and maintain the blockchain, a private blockchain is limited to only specific nodes. As a result, the cryptocurrency market has witnessed a considerable number of altcoins with substantial innovative features.

3.3 Bitcoin Security Issues - Double-Spending Attacks

Double-spending is an attack that could be performed by malicious users attempting to deceive the system by spending the same BTC more than once. The attacker generates duplicates of the same UTXO and uses it as an input in more than one transaction. Differentiating between the duplicated (fraudulent) copies and the original becomes an issue when used in a decentralized system. There is no trusted entity that verifies the legitimacy of the UTXO used as input in a transaction. The inputs of a transaction may consist of unidentifiable fraudulent BTC that have possibly been spent earlier.

The system defends against such attacks by relying on its users (miners) to validate the legitimacy of the BTC used as an input to transactions. Using the information stored in the blockchain from the previous transactions, the miners validate the inputs of any new transaction to ensure that it does not contain previously spent inputs. Once verified, the transaction is mined into a block that is attached to the blockchain. Any user that refers to the blockchain becomes aware that specific UTXO(s) have been spent earlier, making fraudulent input transactions detectable.

To ensure that attackers cannot manipulate the blockchain in their favor, the mining process is designed to be an expensive and resource-intensive operation. To mine a block of transactions in the blockchain, the miners must provide a valid proof-of-work. An attacker that wishes to double-spend BTC must reverse a transaction that has been stored in the blockchain to reuse its inputs in another transaction. Reversing an already stored transaction in the blockchain is an extremely difficult task since it requires a significant share of the total computational power of the system.

In the rest of this section, we will analyze the double-spending attacks. We first discuss conventional methods to perform double-spending. Next, we analyze the probability and profitability of the double-spending attack and present a trade-off between the waiting time before accepting a transaction versus the profitability of the attack.

3.3.1 Types of Attacks

A double-spending attack comes in many forms. We discuss various techniques that can be performed.

3.3.1.1 Race Attack

A race attack refers to the case where a merchant accepts an unconfirmed transaction (a transaction in a transaction pool waiting to be mined and stored in the blockchain) and immediately provides the payer with a product/service before waiting for confirmation. An attacker with the intention of deceiving the merchant creates two transactions: (i) a transaction that pays the merchant an amount of BTC in return for a product/service and (ii) a fraudulent transaction that pays the same amount to the wallet of the attacker. Both transactions use the same inputs (duplicated BTC) and try to spend the same BTC. The attacker concurrently releases both transactions into the Bitcoin network. The miners consider both transactions as being valid until one of them gets stored in the blockchain. At that point, the inputs of the stored transaction cannot be used as inputs to other transactions. Therefore, the fraudulent transaction has a chance of being verified first and added to the blockchain making the merchant-paying transaction invalid. The invalid transaction is rejected by the system and dropped from the transaction pools of miners.

To avoid a race attack, merchants must wait for the mining to be completed and the transaction to appear in the blockchain before providing the payer with the product/service.

It is recommended that the merchant should wait for at least six subsequent blocks as confirmation before making the trade. In this case, the chances for an attacker to reverse a transaction are negligible, assuming that the attacker can control no more than 10% of the total computational power used in mining.

3.3.1.2 Finney Attack

Finney attack was first suggested in a Bitcoin forum [63]. Similar to the race attack, the attacker performing this attack will only succeed if the merchant accepts an unconfirmed transaction. The attacker creates two transactions similar to those in the race attack and holds on to both of them. The attacker then begins mining the block containing the fraudulent transaction. If the attacker is successful in mining the block, the attacker then uses the other transaction to pay a merchant immediately in exchange for a product/service. Once the merchant makes the trade, the attacker releases the mined block that contains the fraudulent transaction into the network. Given that the block is already mined, it will be added to the blockchain immediately. As a result, the merchant-paying transaction will become invalid. In addition to this, the attacker is rewarded the mining reward for the mined block carrying the fraudulent transaction. However, the ability to independently mine a block is improbable given the resources necessary to perform the task.

3.3.1.3 Vector76 Attack

In comparison to the race and Finney attacks, the Vector76 attack requires the merchant to wait for a single block to be mined and added to the blockchain as a confirmation. To reverse the transaction, the attacker needs to create a fork in the blockchain. Initially, the attacker creates a merchant-paying transaction and does not broadcast it to the network.

Next, the attacker tries to independently and secretly mine this transaction into a block. If successful, the attacker holds onto the block until the honest miners discover another block. The attacker then simultaneously releases the block into the network at the same time as the honest miners release their block that will result in a fork. Before the fork is resolved, the attacker creates a fraudulent transaction that double-spends the same BTC used in the merchant-paying transaction. The attacker then relays the fraudulent transaction to the honest miners that do not have the path of the blockchain that carries the merchant-paying transaction. These miners see the fraudulent transaction as valid and begin mining it into a block. As a result, each path of the blockchain stores one of the transactions. If the path that holds the fraudulent transaction grows longer than the other path, the double-spending attempt is successful.

3.3.1.4 51% Attack

51% attack is the largest threat to the BTC system. This attack is also referred to as the majority attack in which the attacker (usually a pool of miners) controls more than half of the total computational power of the system. By controlling the majority of the power, the attacker is capable of interfering with the process of mining blocks and reversing any block of transactions. During a 51% attack, the system loses integrity since the other miners no longer have an incentive to compete in the mining process.

To better comprehend this attack, consider the case where the attacker generates a merchant-paying transaction and releases it into the network. The merchant waits for an appropriate number of confirmations before accepting the payment and making the trade. Simultaneously, the attacker secretly begins to mine a block that contains a fraudulent transaction followed by more blocks to extend it. Since the computational power of the attacker is more than the rest of the computational power of all the miners combined, the attacker can mine blocks in less time. Once the merchant accepts the transaction, the attacker releases the secretly mined blocks to create a fork in the blockchain. If the fraudulent fork created by the attacker is longer than the original chain, it becomes dominant and all miners begin to extend on it. By that, the merchant-paying transaction no longer exists in the blockchain.

This attack represents the biggest threat to Bitcoin as it is directly correlated to the resources an attacker can provide. Resources are measured in terms of financial and computational power. Large entities such as governments or intelligence agencies have the means to control a large share of the total computational power. They are able to destroy or push the system into their favorable status. It is important to note that even with a computational power that is slightly less than 50%, an attacker may still be able to severely manipulate the system. In the next subsection, we analyze the chances of success of the attackers based on the share of computational power they control.

3.3.2 Probability of Success

Despite the continuously increasing popularity of Bitcoin, the number of merchants that have accepted it as a method of payment today is still relatively minimal. Many merchants have concerns about its capabilities in terms of security, while others consider it as a slow method to make payments. Those that accept it should try to take all precautions before accepting a transaction to prevent double-spending attacks.

One of the important precautions is to decide when to accept a transaction before making the trade. Merchants prefer to obtain a certain degree of confidence as assurance that the payer will not be able to reverse the transaction. Those that can afford to wait a long period before accepting a transaction (for example, online platforms) require a minimum of six confirmations before accepting a transaction and considering it as being irreversible. However, others that cannot afford this time waiting (such as vending machines), rush into accepting transactions at the risk of losing the payment to a double-spending attack.

Similar to the analysis in [51], we model the race between the honest miners and the attacker to generate blocks as a binomial random walk. The race is denoted as z that represents the number of blocks generated by the honest miners with computational power p minus the number of blocks generated by the attacker with computational power $q = 1 - p$. If a block is generated by the honest miners, we increment z by 1. Conversely, if a block is generated by the attacker, we decrement z by 1. The race between the honest chain and the chain generated by the attacker can be derived as

$$z_{i+1} = \begin{cases} z_i + 1, & \text{with probability } p, \\ z_i - 1, & \text{with probability } q, \end{cases} \tag{3.11}$$

where i represents an individual block race. If $q > p$ and the attacker has unlimited resources, the attacker will eventually reach $z < 0$. At that point, the attacker can replace the blocks generated by the honest miners and succeed in performing the attack.

The probability of the attackers to catch up and surpass the blocks generated by the honest miners can be compared to the Gambler's Ruin Problem. Similar to the description in [64], we assume a gambler (attacker) begins with an initial fortune i, $0 < i < N$, and either wins \$1 with probability q or loses \$1 with probability $p = 1 - q$, in each successive gamble. The game represents a random walk that terminates at $i = 0$ (fail) or at $i = N$

(success). The probability of success after i trials is denoted as P_i and can be calculated as:

$$P_i = qP_{i+1} + pP_{i-1}. \tag{3.12}$$

Since $q + p = 1$, we can rewrite equation (3.12) as:

$$P_{i+1} - P_i = \frac{p}{q}\left(P_i - P_{i-1}\right). \tag{3.13}$$

At $i = 0$, the attacker has a probability of success $P_0 = 0$. By rearranging and generalizing equation (3.13), we have

$$\begin{aligned} P_{i+1} &= P_1 \sum_{j=0}^{i} \left(\frac{p}{q}\right)^j \\ &= \begin{cases} P_1 \frac{1-(\frac{p}{q})^{i+1}}{1-(\frac{p}{q})}, & \text{if } p \neq q, \\ P_1(i+1), & \text{if } p = q = 0.5. \end{cases} \end{aligned} \tag{3.14}$$

Let $i = N - 1$ meaning that $P_N = 1$, we can rewrite equation (3.14) as

$$1 = P_N = \begin{cases} P_1 \frac{1-(\frac{p}{q})^N}{1-(\frac{p}{q})}, & \text{if } p \neq q, \\ P_1 N, & \text{if } p = q = 0.5. \end{cases} \tag{3.15}$$

Solve P_1 from equation (3.15) and substitute the result into equation (3.14) to obtain

$$P_i = \begin{cases} \frac{1-(\frac{p}{q})^i}{1-(\frac{p}{q})^N}, & \text{if } p \neq q, \\ \frac{i}{N}, & \text{if } p = q = 0.5. \end{cases} \tag{3.16}$$

Following the analysis in [65], we assume that the attacker begins with an initial fortune $i = y$ and can afford to lose up to y dollars before giving up. The gambler wins if $i = N = y + z + 1$ dollars. This assumption modifies the game to account for the probability P_s of the attacker to surpass the blocks generated by the honest miners as

$$P_i = \begin{cases} \frac{1-(\frac{p}{q})^y}{1-(\frac{p}{q})^{y+z+1}}, & \text{if } p \neq q, \\ \\ \frac{y}{y+z+1}, & \text{if } p = q = 0.5. \end{cases} \tag{3.17}$$

Consider an attacker that possesses an unlimited amount of resources and is willing to use as much of it as needed to perform the attack, i.e. $y \to \infty$. If $q > p$, then

$$\lim_{y\to\infty} \frac{1-(\frac{p}{q})^y}{1-(\frac{p}{q})^{y+z+1}} = 1. \tag{3.18}$$

For $q < p$, we first divide the numerator and denominator by $\left(\frac{p}{q}\right)^y$ then calculate the limit as

$$\lim_{y\to\infty} \frac{(\frac{p}{q})^{-y} - 1}{(\frac{p}{q})^{-y} - (\frac{p}{q})^{z+1}} = \left(\frac{q}{p}\right)^{z+1}. \tag{3.19}$$

Finally, we can summarize the probability of the attacker to surpass the blocks generated by the honest miners as

$$Q_z = \begin{cases} \left(\frac{q}{p}\right)^{z+1}, & \text{if } q < p \text{ or } z \geq 0, \\ \\ 1, & \text{if } q > p \text{ or } z < 0. \end{cases} \tag{3.20}$$

The merchant has no way of figuring out the number of blocks that the attacker has been able to secretly mine. Therefore, one way to model the overall probability of the attacker to surpass the honest chain is by using the Poisson distribution. The expected number of blocks an attacker can generate is $\lambda = (z+1)\left(\frac{q}{p}\right)$. The overall probability P_s of the attacker to surpass the honest chain can be computed by multiplying the Poisson density and the probability of surpassing the honest $z-k$ remaining blocks as discussed in equation

$$\begin{aligned} P_s &= \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \times Q_{z-k} \\ &= 1 - \sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \times \begin{cases} 1 - \left(\frac{q}{p}\right)^{z-k+1}, & \text{if } q < p \text{ or } k \leq z, \\ 1 - 1 = 0, & \text{if } q > p \text{ or } k > z. \end{cases} \end{aligned} \tag{3.21}$$

For equation (3.21), if $q > p$, we will always have $P_s = 1$, meaning that the attacker will win. When $q < p$, the probability for the attacker to succeed is

$$P_s = 1 - \sum_{k=0}^{z+1} \frac{\lambda^k e^{-\lambda}}{k!} \times \left(1 - \left(\frac{q}{p}\right)^{z-k+1}\right). \tag{3.22}$$

Another way to model this probability is by using the negative binomial distribution assuming the attacker can pre-mine one block before broadcasting the merchant-paying transaction to the network [66]. The merchant waits for n blocks to be generated by the honest miners with computational power p before accepting the transaction. At that time, the attacker can secretly generate m blocks with computational power $q = 1 - p$, where $m = n - z - 1$. By definition, we can model this as the m number of blocks that the attacker can generate (success) before the n number of blocks the honest miners can generate (failure). Therefore, the probability of a successful double-spending attack for a given value m can be calculated as

$$P(m) = \binom{m+n-1}{m} \times p^n q^m. \tag{3.23}$$

Overall, the probability for an attacker to successfully surpass the number of blocks generated by the honest miners can be computed as

$$\begin{aligned} P_s &= \sum_{m=0}^{\infty} P(m) \times Q_{n-m-1} \\ &= 1 - \sum_{m=0}^{\infty} \binom{m+n-1}{m} \times p^n q^m \times \begin{cases} 1 - \left(\frac{q}{p}\right)^{n-m}, & \text{if } q < p \text{ or } k \leq n-m, \\ 1 - 1 = 0, & \text{if } q > p \text{ or } k > n-m. \end{cases} \end{aligned} \tag{3.24}$$

Similar to the previous analysis, equation (3.24) confirms that when $q > p$, the attacker will always succeed since $P_s = 1$. When $q < p$, the probability of success can be defined as

$$\begin{aligned} P_s &= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{k} \times p^n q^m \times \left(1 - \left(\frac{q}{p}\right)^{n-m}\right) \\ &= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{m} \times (p^n q^m - p^m q^n). \end{aligned} \tag{3.25}$$

Fig. 3.3 shows the results of P_s as n changes based on equation (3.25). From this figure, the merchant can obtain the desired level of confidence before accepting a transaction. The obtained level of confidence is definite for any q at $n = 0$, meaning that the attackers have a 100% chance of success. As the number of blocks n increases, the chances of a successful double-spending attack decline. Conversely, as q increases, the chances of a successful attack increase. The figure also shows that if $q \geq 0.5$ then we will always get $P_s = 1$. This is known as the majority attack. In fact, even if the values of q are slightly less than 0.5, the chances of a successful double-spending attack could still be high. However, the probability declines exponentially as the value of n increases.

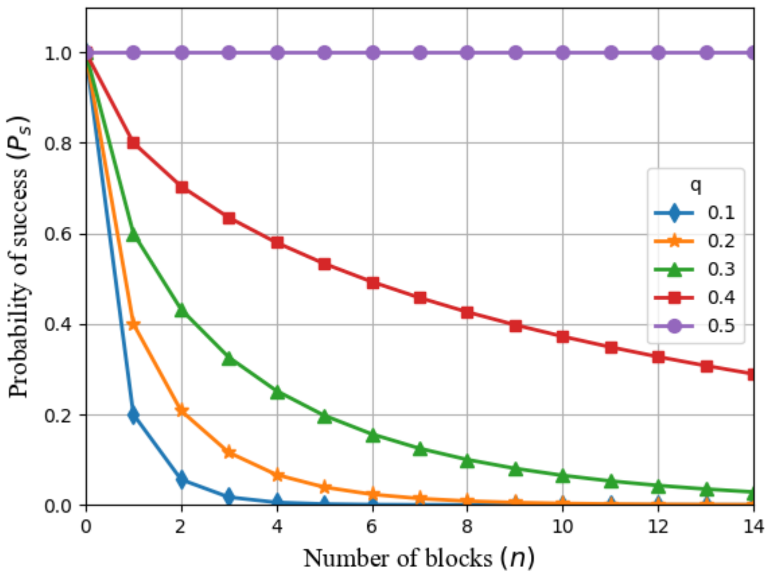


Figure 3.3: Probability of successful double-spending attacks vs. number of confirmations waited by the merchant.

3.3.3 Attack Profitability

A successful double-spending attack is only profitable if the revenue is higher than the cost of performing the attack. Suppose an attacker tries to double-spend v BTC paid to a merchant in exchange for a product/service. The attacker releases a transaction into the network that pays v BTC to the wallet possessed by the merchant. Immediately after releasing the transaction, the attacker secretly begins to mine blocks of transactions. One of these blocks contains a fraudulent transaction that pays the same v BTC to the wallet possessed by the attacker. The merchant accepts the transaction after observing that n blocks have been extended to the blockchain. If the attacker is able to secretly mine $m = n + 1$ blocks and replace the n blocks in the blockchain generated by the honest miners, then the attacker is successful in gaining a product/service without paying for it. Assume that the attack returns a value of $2v$, one of which is the actual BTC as a result of reversing the merchant-paying transaction and the other as the product/service. In addition to this, the attacker gains the mining reward for each block mined and the transaction fees included in each transaction. Then the revenue gained by the attacker can be formulated based on his/her corresponding P_s as follows

$$\mathsf{Revenue} \approx v + P_s(v + Rm) \text{ BTC}, \tag{3.26}$$

where R is the block reward and the transaction fee per block.

Multiple factors can impact the cost such as the price and depreciation value of machinery used, the cost of electricity, and the amount of BTC being spent in the transaction. However, formulating the cost with all the possible factors is infeasible. To simplify it, we focus our analysis on the cost factors that could change significantly as the attack is performed. These factors include the v BTC an attacker spends in the merchant-paying transaction, the cost of mining m blocks, and the depreciation cost $d(t)$ of the computing device used in BTC at time t. We derive the cost as follows

$$\mathsf{Cost} \approx v + me_q(t) + d(t) \text{ BTC} \tag{3.27}$$

where $e_q(t)$ is the estimated mining electrical cost in BTC/block of a miner with a share q of the total computational power of the system. We assume $e_q(t)$ remains constant during the total time T the attack is performed.

We also assume that the average lifespan of the mining equipment is approximately two years. Using straight-line depreciation, $d(t)$ is a negligible value for an attack over a short period. Therefore, we can reduce the cost equation as follows

$$\mathsf{Cost} \approx v + me_q(t) \text{ BTC} \tag{3.28}$$

The time to mine a single block either by the honest miners or the attackers is approximately ten minutes. Therefore, we can rewrite equations (3.25), (3.26), and (3.28) as

$$P_s \approx 1 - \sum_{m=0}^{\frac{T}{10}-1} \binom{m + \frac{T}{10} - 1}{m} \times (p^{\frac{T}{10}} q^m - p^m q^{\frac{T}{10}}), \tag{3.29}$$

$$\textsf{Revenue} \approx v + P_s \left(v + \frac{RT}{10} \right) \text{ BTC}, \tag{3.30}$$

$$\textsf{Cost} \approx v + \left(\frac{T}{10} \right) e_q(t) \text{ BTC}. \tag{3.31}$$

The profit/loss can be formulated from equations (3.30) and (3.31) as

$$\begin{aligned} \textsf{Profit/Loss} &= \textsf{Revenue} - \textsf{Cost} \\ &\approx P_s \left(v + \frac{RT}{10} \right) - \left(\frac{T}{10} \right) e_q(t) \text{ BTC}. \end{aligned} \tag{3.32}$$

Nowadays, to stand a chance in mining Bitcoin, miners merge their computational power into mining pools as discussed in Section 3.2.7. The mining pools combine the computational power provided by the computing machine of each participating miner. Machines are categorized into one of four groups: Application-Specific Integrated Circuits (ASIC), Field-Programmable Gate Array (FPGA), Graphics Processing Unit (GPU), and Central Processing Unit (CPU). Each group can provide up to a certain computational power. Comparisons of most of those machines are presented in [67] and [68].

Each computing machine consumes electricity differently based on its specifications. Even machines with similar specifications might vary in cost to operate. As a result, formulating the cost of electricity spent by a miner in the mining process becomes challenging.

ASICs have monopolized the mining process due to their incomparable computational power with those of the CPUs, GPUs, and FPGAs. Miners using any computing machines other than ASICs have a negligible chance of competing. Many mining pools do not permit miners with these machines to join their pools. A miner that joins a mining pool with one of these machines would hardly earn BTC in the event that the pool successfully mines a block. This is because rewards are usually divided among the miners based on their contributions as discussed in Section 3.2.7.

Our goal now is to formulate the estimated electrical cost $e_q(t)$ of a mining pool. First we estimate the total number of miners $N(t)$ based on the total hashrate $H(t)$ of the system at a certain time t as

$$N(t) \approx \frac{H(t)}{h(t)}, \tag{3.33}$$

where $h(t)$ is the average hashrate of a single mining machine involved in mining at time t.

The cost of electricity is measured in cents/kWh and varies based on the end-use sector and time t. End-use sectors include, residential, commercial, industrial, and transportation. We denote the average cost of electricity of all sectors at time t as $e_a(t)$. Using the computing wattage w of the machine, the average running cost $c(t)$ of a machine at time t is

$$c(t) \approx e_a(t) \times w \text{ cents/hour.} \tag{3.34}$$

Using equations (3.33) and (3.34), the total cost $E(t)$ for all miners at time t is

$$E(t) \approx N(t) \times c(t) \text{ cents/hour.} \tag{3.35}$$

We know that it takes approximately 10 minutes to generate one block, i.e. in $T = 1$ hour, miners can generate $m = 6$ blocks. Therefore, the total cost $C(t)$ for all miners to generate one block at $T = 10$ minutes can be estimated as

$$C(t) \approx \frac{E(t)}{6} \text{ cents/10 minutes (1 block).} \tag{3.36}$$

However, as discussed previously, miners merge their computational power to increase their chances of winning in the mining competition. The largest mining pool that exists today is Antpool [69] controlling approximately 25% of the total computational power. Other mining pools also exist such as BTCC Pool [70], Bixin [71], BTC.com [72], and BTC.TOP [73] that control approximately 7%-11% of the computational power.

We estimate the average electricity cost $e_q(t)$ of a mining pool based on its computational power q as

$$\begin{aligned} e_q(t) &\approx C(t) \times q \\ &\approx \frac{H(t) \times e_a(t) \times w \times q}{6h(t)} \text{ cents/block.} \end{aligned} \tag{3.37}$$

For our simulations, we assume that the total cost of mining blocks $C(t)$ by all miners and computational power q of the mining pool are fixed during the total mining time T. We also assume a mining environment consists of miners using only ASICs such as Antminer S9 since it is one of the most efficient computing machines on the market today. The specifications of this machine are $h = 14$ TH/s and $w = 1.375$ kWh.

Consider an attacker trying to perform a double-spending attack during October 2019. During that period, 1 BTC was equal to approximately \$9,200. The total hashrate power

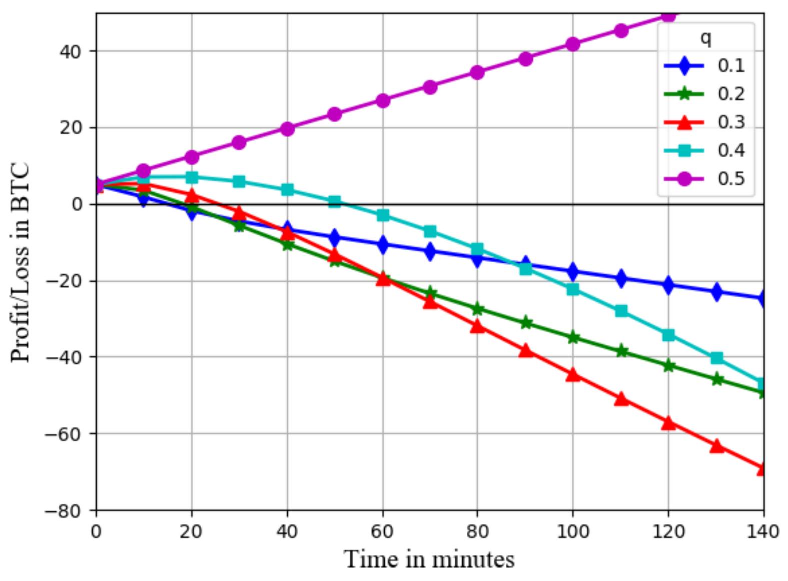


Figure 3.4: Profits/losses of attackers with varying computational power q trying to double-spend $v = 5$ BTC during October 2019.

was approximately $H = 95000000$ TH/s and the average cost of electricity for all sectors in the U.S. was approximately 10.45 cents/kWh, based on the data collected by the U.S Energy Information Administration [74]. Under these circumstances, in Fig 3.4 we present the expected profit/loss of double-spending attacks for various computational powers q. For this analysis, we assume the attackers try to double-spend $v = 5$ BTC.

In Fig. 3.4, a point above $y = 0$ represents a profit while one below it represents a loss. The point of intersection of a curve with $y = 0$ represents the break-even point of an attack. The amount of BTC spent to perform the attack at this time is equal to the revenue returned. By analyzing the figure, we attain the following findings:

1. For all values q at $t = 0$, the attacker turns a profit of exactly 5 BTC. As shown in Fig. 3.3, for any value q at $n = 0$ (or $t = 0$), $P_s = 1$. This may occur when the receiving user accepts an unconfirmed transaction where the attacker has a theoretically perfect chance to succeed.

2. If the receiving user waits for n confirmations before accepting a transaction, the attacker must compete to mine blocks for the blockchain. Based on the previous analysis, we know that the probability of success P_s of mining blocks is based on the computational power q of the attacker. We also know that P_s declines as n (or t) increases for all values $q < 0.5$. Therefore, not only does a profit turn into a loss as the attack time progresses, but an attacker with a smaller q will more likely lose at an earlier point in time. However, with smaller values q, losses are also smaller. This is evident as larger values q impose higher electricity costs $e_q(t)$.

3. Looking closer at the attack with $q = 0.1$, we notice that the attacker breaks-even after approximately 15 minutes, i.e. after mining at most one block. Beyond that time, if the attacker has not been able to fork the block, she will begin losing if she continues to perform the attack. This means, the attacker would most likely surrender at that time to avoid losses.

4. For $q = 0.2$ to $q = 0.4$, the potential profits first grow as t increases and rewards are accumulated until they reach a turning point where they begin to decline and eventually turn into losses. This turning point occurs when P_s starts to decline with t.

5. For $q \geq 0.5$, the attacker will always turn a profit representing the majority attack. This is evident as represented by the straight line with a continuous positive slope.

In summary, an attacker with a computational power $q < 0.5$ will eventually lose at some point as t increases. On the other hand, an attacker with computational power $q \geq 0.5$ will always succeed with a profit. However, it is important to note that this analysis does not include the *luck* factor. Consider two miners with computational powers q_1 and q_2 respectively, where $q_1 > q_2$. The miner with computational power q_1 has more resources to solve the proof-of-work, therefore can perform mining faster than the miner with computational power q_2. However, the miner with computational power q_2 could still find the solution to the proof-of-work first due to the randomness of the exhaustive search performed. From a probabilistic standpoint, the chances are low.

3.4 Summary

Since the inception of Bitcoin, the popularity of the blockchain technology has grown aggressively. In this chapter, we first introduced the background of Bitcoin and explicated its major building blocks and protocols. Next, we delved into crucial security concerns. We began by discussing the double-spending attacks and analyzing their probability of success. Using this analysis, we further evaluated the profitability of the attack. We showed that attackers with less than half of the total computational power of the system will eventually lose at some point while performing the attack.

Chapter 4

d-MABE: ***Distributed*** Multilevel Attribute-Based EMR Management and Applications

4.1 Introduction

After introducing the major concepts of blockchain and extensively looking into its security, in this chapter, we propose a novel distributed data sharing scheme that leverages blockchain and smart contracts. We choose electronic medical records as our first application to demonstrate the benefits of alleviating dependence on the record-generating institutions and thus empowering patients over their own sensitive data. Our proposed work aims at solving the security and privacy challenges and interoperability issue of the current solutions. In the existing systems, EMRs are stored within the generating institution, which significantly limits the patients' access of their own records. They must also trust that the institutions will not leak any sensitive information of their EMRs to unintended organizations. In comparison, our proposed work empowers them over their EMRs. First, it provides record ownership to the patients, allowing them to be in actual possession of their records, rather than have them stored by the generating institution. Second, it allows patients to selectively share different parts of their records with distinct data users based on their privacy preferences. Using our proposed scheme, the record-generating institutions can be eliminated from the record-sharing process, hence, eliminating the need for the third trust party.

The rest of this chapter is organized as follows. In Section 4.2 the problem formulation is described outlining the system model and our design goals. Next, in Section 4.3, our proposed scheme is presented in detail outlining the proposed algorithms. In Section 4.4 we formally prove the security and privacy of our proposed scheme. Following that, in Section 4.5, we present a performance and application analysis that is supported with numerical results in Section 4.6. Finally, in Section 4.7 , we provide an extended application discussion and draw a conclusion in Section 4.8.

4.2 Problem Formulation

Upon visiting a medical institution, a patient interacts with a wide spectrum of distinct staff members that may include but is not limited to: admittance staff members, physicians assisted by nurses and technicians that provide the required treatment, and maybe even financial or discharging staff members. If those staff members can individually and efficiently access the necessary parts of previous medical record(s) of the patient generated by other institutions required to perform their specific jobs, they can provide the patient with expedited admittance and enhanced diagnosis and treatment decisions, reducing the overall time spent during the visit. For example, an ER physician may learn severe drug allergies or current medication for an unresponsive patient being rushed into the ER from his/her previous records. As a result, the physician can prevent iatrogenic illnesses caused by administering inappropriate medication to that patient or harmful drug interactions. By accessing critical medical information, patient safety is enhanced while saving time and resources. However, for the sake of patient privacy, only necessary staff members should be given permission to access patient record(s) from the medical history determined by the profile of the patient.

In this chapter, we denote a record generated by an institution for a patient as $\mathcal{R}$. We denote the record attributes as $\{\mathsf{R}_j \in \mathcal{R} \mid 1 \leq j \leq n\}$, where n is the maximum number of attributes within the record. Record attributes may include previous medical history, medical treatment, lab tests, medication, personal information, financial statements, credit card information, and others. The corresponding attribute values of the record generated for the patient can be denoted as $\{a(\mathsf{R}_j) \mid 1 \leq j \leq n\}$.

In current systems, records often have a variety of attributes and may be blocked or intentionally delayed by competitive record-generating institutions. To prevent this, record-generating institutions should have minimal control over the records of patients whereby empowering patients over their own records. Patients may have different attitudes in terms of sensitivity of their record attributes. The challenge today is to provide patients with a method that allows them to share the attribute values of their records efficiently and securely while maintaining the privacy preferences they desire. Patients should also be able to selectively share certain parts of their record(s) with the healthcare providers they select. Although patients may not easily understand how to share the medical parts of their record(s), empowering them would at least allow them to consult their trusted medical staff such as their Primary Care Physicians (PCPs) to decide on how and which parts of their medical record(s) are to be shared. With their consent, patients may even delegate this process to such entities to avoid situations such as being unconscious, requiring someone to make decisions for them. Lastly, patients should not be concerned with the availability nor the guaranteed secure storage of their records.

4.2.1 Design Goals

Our proposed scheme aims at satisfying the following:

Data Ownership: Patients are empowered over their records. They have full ownership and control over how to share their records.

Fine-grained access control: Patients can grant specific access permissions to the desired staff members based on their privacy preferences. They can selectively share any part of their records using any set of staff member attributes they wish, limiting access to specific parts of their record and certain staff members at particular medical institutions.

Collusion resistant: Two or more staff members that possess different sets of attributes or are employed at the same/different institution(s) cannot combine their attributes to gain access to any part of the records they are not authorized to access individually.

Data confidentiality: Records are completely protected from any unauthorized staff members that do not possess the correct set of attributes predefined by the patient. This includes any type of space that stores the records for the patients.

Distributed storage: Patient records are stored in a distributed storage network. They can be accessed at any time and by any permissioned staff member. Storage is not centralized or controlled by the record-generating institution.

Data access trace-ability: Patients can trace-back to the entire history of accesses of their records. This allows patients to keep track of how their records are being accessed.

4.2.2 System Model

The general model of our proposed record-sharing scheme is illustrated in Fig. 4.1. The system consists of six main entities:

Patients: Patients are data owners that wish to share their data selectively based on their privacy preferences. We denote the set of patients as $\{p_a \in \mathcal{P} \mid 1 \leq a \leq \infty\}$.

Medical institutions: Medical institutions provide medical treatment and generate medical records for the patients. We denote the set of medical institutions as $\{m_b \in \mathcal{M} \mid 1 \leq b \leq \infty\}$.

Staff members: personnel employed at a medical institution. We denote the set of staff members at medical institution m_b as $\{s_{b,l} \; \forall \; 1 \leq l \leq \infty\}$. Staff members are categorized into groups of similar characteristics based on the set of attributes $\mathbb{A} = \{\mathsf{A}_1, \mathsf{A}_2 \ldots, \mathsf{A}_n\}$ they each possess. Attributes represent identifiers that are selected from an infinite pool set. They may be as general as the role of a staff member and could be as unique as a biometric.

Key-issuer: a semi-trusted party that generates keys for permissioned staff members upon their requests to access parts of the record.

Distributed storage P2P network: a non-trusted P2P network used by the patients to store and share their records with staff members at any medical institution.

Blockchain P2P network: a non-trusted P2P network that maintains a blockchain to regulate access permissions of patient records to the staff members of medical institutions.

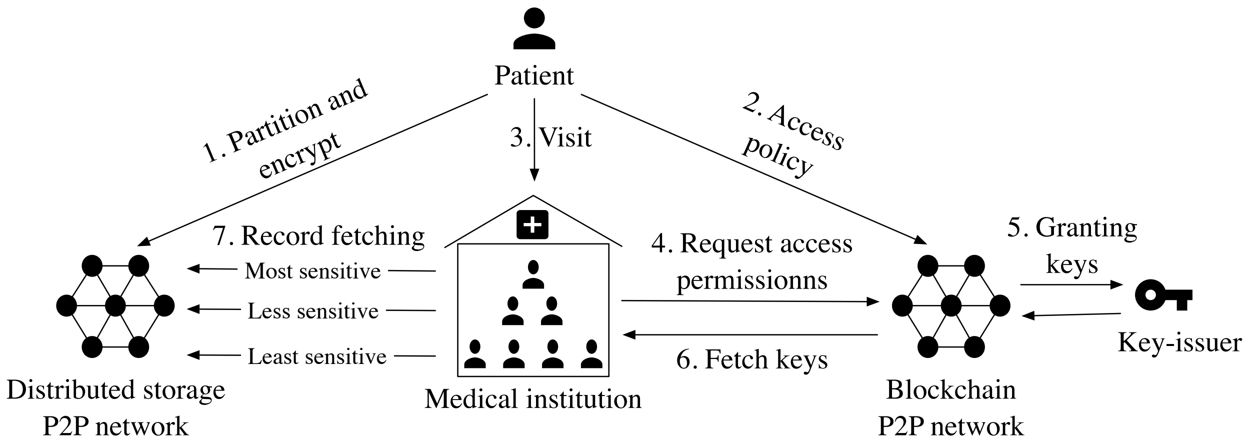


Figure 4.1: d-MABE general scheme orchestration.

In general, in order for patients, medical institutions, staff members or key-issuers to interact with any of the two P2P networks, they must run nodes that are capable of connecting to either network. These nodes may be light-weight nodes (similar to the simple payment verification nodes in Bitcoin [51]) that can assemble transactions and propagate them to the network. Running light-weight clients does not require powerful computing machines and can be done via simple machines such as mobile devices making our proposed scheme accessible to everyone. The common requirement for running either node is being able to generate the public key Pub and private key Pr pair.

4.3 The Proposed d-MABE Scheme

In this section, we first present an overview of the major chain of events in our proposed scheme. For discussion purposes, we assume that a patient p_a has already received some medical treatment at institution m_b and that a medical record $\mathcal{R}$ was generated for him/her. We also assume that p_a anticipates visiting some other institution $\{m_b \mid b \geq 1\}$ in the future and would like to share $\mathcal{R}$ selectively among its staff members. Based on our description, we then propose a generalized structure through three smart contracts that can be deployed on a blockchain.

4.3.1 Scheme Orchestration

The proposed scheme can be divided into seven major events as demonstrated in Fig. 4.1.

Record partition and encryption: Following the work presented in [75], p_a initially defines a privilege-based access structure that satisfies his/her privacy desires. The access structure is divided into k levels $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k\}$ where each level is associated with an access policy $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$. The p_a then partitions $\mathcal{R}$ into k segments $\{\mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_k\}$ of different sensitivity. Each $\mathsf{R}_i \in \mathcal{R}$ may represent one or more record attribute(s) depending on how p_a partitions $\mathcal{R}$. Next, p_a derives a set of symmetric keys $\{\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_k\}$ as described in Section 2.3.1. Using these keys, p_a then encrypts each corresponding $\mathsf{R}_i \in \mathcal{R}$ as

$$\mathsf{ER}_i = \mathsf{Sym\text{-}Enc}_{\mathsf{sk}_i \| \mathsf{SBK}}(\mathsf{R}_i), \tag{4.1}$$

where $\mathsf{Sym\text{-}Enc}$ is a symmetric encryption algorithm such as the Advanced Encryption Algorithm (AES) [44] and SBK denotes the subscription based key that is only accessible to active members through membership subscription authentication. Members have no direct access to SBK and can only use it while on site. This design serves two purposes: (i) it prevents any users from sharing the $\mathsf{ER}_i's$ even if they share their sk_i's with others and give them unprivileged access and (ii) it provides an easy option to disable unsubscribed members from accessing EMRs. In other words, with this design, key revocation is no longer needed. Finally, using the $\mathsf{Encryption}$ function discussed in Section 2.3.3, p_a encrypts each sk_i under its corresponding $\mathcal{T}_i$ defined in the privilege-based access structure, such that

$$\mathsf{Esk}_i = \mathsf{CPABE\text{-}Enc}(\mathsf{PK}, \mathsf{sk}_i, \mathcal{T}_i), \tag{4.2}$$

where $\mathsf{CPABE\text{-}Enc}$ is the CP-ABE encryption function and PK is the public key generated during the Setup phase. The p_a then stores the generated ciphertexts $\{\mathsf{ER}_1, \mathsf{ER}_2, \ldots, \mathsf{ER}_k\}$ and $\{\mathsf{Esk}_1, \mathsf{Esk}_2, \ldots, \mathsf{Esk}_k\}$ over IPFS. In cases that EMRs consist of multiple/diverse attributes, it may be a burden for patients to define their privilege-based access structures and may even leak sensitive information if defined inappropriately. A possible approach that patients may consider is to divide their records into multiple categories based on the nature of data being shared. For example, a record could be divided into personal information, medical information, and financial information. For each category, a patient can then define a separate privilege-based access structure which would reduce the overall complexity of each structure. Patients may even consult their trusted Primary Care Physicians (PCPs) on how to define a privilege-based access structure for proper medical information sharing.

Access policy: To facilitate data management and sharing, p_a shares the privilege-based access structure that incorporates the access policies $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$. The access structure is incorporated into a smart contract that is then deployed over the blockchain.

Medical institution visit: When p_a visits a medical institution $\{m_b \mid b \geq 1\}$ to get medical treatment, p_a interacts with various staff members that may or may not belong to a predefined privilege-based access structure.

Requesting access permissions: Permissioned staff members will be granted access to certain parts of a record based on the attributes they possess. To request access, a staff member $s_{b,l}$ interacts with the smart contract previously deployed by p_a that incorporates the privilege-based access structure. The smart contract will verify whether the $s_{b,l}$ possesses a set of attributes that can satisfy an access policy within the access structure. Once the verification is performed the smart contract will publicly announce the access permissions of the $s_{b,l}$ over the blockchain.

Granting keys: The key-issuer continuously monitors the blockchain for access announcements. An announcement contains the set of attributes $\mathbb{A}$ that the $s_{b,l}$ possesses. It is a form of verification to the key-issuer that the $s_{b,l}$ possesses set $\mathbb{A}$ before generating a secret key SK using the $\mathsf{KeyGeneration}$ function described in Section 2.3.3. The key-issuer then encrypts SK with the public key Pub of $s_{b,l}$ as the following

$$\mathsf{ESK} = \mathsf{Asym\text{-}Enc}_{\mathsf{Pub}}(\mathsf{SK}), \tag{4.3}$$

where $\mathsf{Asym\text{-}Enc}$ is the asymmetric encryption function. To share the key with the $s_{b,l}$, the key-issuer transacts with a global smart contract that publicly announces the encrypted key over the blockchain.

Key fetching: The $s_{b,l}$ can obtain the uniquely encrypted secret key ESK by monitoring the announcements made over the blockchain. Once obtained, the $s_{b,l}$ can decrypt ESK using his/her private key Pr that corresponds to the public key Pub such that

$$\mathsf{SK} = \mathsf{Asym\text{-}Dec}_{\mathsf{Pr}}(\mathsf{ESK}), \tag{4.4}$$

where $\mathsf{Asym\text{-}Dec}$ is the asymmetric decryption function.

Record fetching: The storage location of the encrypted EMRs over IPFS is also provided to the $s_{b,l}$ in an encrypted form under his/her public key in the announcement made over blockchain. The $s_{b,l}$ decrypts the IPFS location to fetch the EMRs. Here, the $s_{b,l}$ possesses the necessary key to decrypt the parts he/she has been granted access to. Once fetched, the $s_{b,l}$ can decrypt the encrypted symmetric key Esk_i using the derived key SK as explained in Section 2.3.3. That is

$$\mathsf{sk}_i = \mathsf{CPABE\text{-}Dec}(\mathsf{Esk}_i, \mathsf{SK}), \tag{4.5}$$

where $\mathsf{CPABE\text{-}Dec}$ is the CP-ABE decryption function. After obtaining sk_i, the $s_{b,l}$ can derive the remaining set of secret keys $\{\mathsf{sk}_{i+1}, \ldots, \mathsf{sk}_k\}$ using equation (2.2). Finally, the $s_{b,l}$ can decrypt each encrypted partition, such that

$$\mathsf{R}_i = \mathsf{Sym\text{-}Dec}_{\mathsf{sk}_i \| \mathsf{SBK}}(\mathsf{ER}_i), \tag{4.6}$$

where $\mathsf{Sym\text{-}Dec}$ is the decryption function.

4.3.2 Smart Contracts

Our proposed scheme includes three main smart contracts: (i) staff member registration, (ii) access verification and permission announcements, and (iii) granting keys.

Staff member registration: Staff Member Registration (SMR) is a global smart contract that registers staff members of medical institutions for patient EMRs access. The process of registering staff members by interacting with this smart contract can be delegated to a

Algorithm 1 SMR smart contract

```
 1: struct staffMember contains
 2:     staffID
 3:     staffAttributes[upBound]
 4: end
 5: variable mapping(address $\rightarrow$ staffMember) Map
 6: function addStaffMember(_address, _id, _attributes[upBound])
 7:     if (msg.sender $\notin$ setOfCertifiers) then
 8:         throw
 9:     else
10:         Map(_address $\rightarrow$ _id, _attributes[upBound])
11:     end if
12: end function
13: function getAttributes(_address)
14:     return Map[_address].staffAttributes
15: end function
```

number of certified institutions that manually verify staff members against their possessed attributes before registering them. This is achieved by incorporating precise policies into the smart contract which requires certain identities to trigger it. Once a certified institution verifies the attributes of a staff member, it executes the smart contract and uses the attributes as input. This results in the attributes being stored over the blockchain. This process maps the identity (in our case the Ethereum address) of the staff member to the set of attributes possessed. By observing the blockchain, we can trace-back the on-going activity of these certified institutions as they add, delete, or modify the information of staff members, hence, we can also detect malicious data manipulation.

Algorithm 1 outlines the general functions of the SMR smart contract. Lines 1-4 represent a generalized data structure staffMember that the smart contract uses to store new staff members. It incorporates the identity staffID and the array of attributes staffAttributes of the staff member. Lines 6-15 represent the two main functions addStaffMember and getAttributes of the smart contract. The addStaffMember function allows only certified insti-

Algorithm 2 AVPA smart contract

```
1: **Initialized variables** _address = SMR address
2: **event** LogAnnounce(address, attributes, $\mathcal{T}_i$)
3: **function** verifyRequest(_msg, $\sigma$)
4:     signer $\leftarrow$ ecrecover(_msg, $\sigma$)
5:     **if** (signer $\neq$ Hash(Pub)) **then**
6:         **throw**
7:     **else**
8:         r $\leftarrow$ SMR(_address)
9:         fetched $\leftarrow$ r.getAttributes(signer)
10:        **for** $i \leftarrow 1$ **to** $k$ **do**
11:            **if** (satisfy($\mathcal{T}_i$, fetched) == $true$) **then**
12:                LogAnnounce(signer, fetched, $\mathcal{T}_i$)
13:                **break**
14:            **end if**
15:        **end for**
16:    **end if**
17: **end function**
```

tutions setOfCertifiers to upload the data of new staff members after physically verifying their attributes. This conditioned access is to prevent malicious attackers from granting access to themselves or others. On the contrary, the getAttributes function can be called by any user. Given the address _address of a specific staff member, this function returns the previously verified and stored attributes of this staff member.

Access verification and permission announcements: The Access Verification and Permission Announcements (AVPA) is a unique smart contract defined by each patient that incorporates his/her personally defined privilege-based access structure in order to facilitate the selective record management and sharing. The staff members interested in obtaining parts of the record interact with AVPA contracts to request access permissions. The smart contract is outlined in Algorithm 2.

The smart contract performs a sequential verification process. This is represented in a single function verifyRequest that takes two inputs: a message _msg prior to being signed and its signature σ. That is

$$\text{\_msg} = \mathsf{Hash}(\mathsf{staffID}), \tag{4.7}$$

$$\sigma = \mathsf{Sign}(\mathsf{Pr}, \mathsf{Pub}, \text{\_msg}), \tag{4.8}$$

where Hash is the hashing function and Sign is an ECDSA [76] signing function.

In the initial verification process, the AVPA smart contract uses an ECDSA compatible validation function $\mathsf{ecrecover}(\text{\_msg}, \sigma)$ to validate σ by verifying whether

$$\mathsf{ecrecover}(\text{\_msg}, \sigma) = \mathsf{Hash}(\mathsf{Pub}) \tag{4.9}$$

holds true. In fact, if the validation function is conducted truthfully, then the correctness of equation (4.9) follows from the ECDSA. Next, signer is compared to the address of the requesting staff member as shown in line 5. If the addresses match, the contract uses signer to fetch the previously verified and stored attributes of this staff member in the SMR contract. However, this method is liable to replay attacks [76]. In section 4.4, we discuss this issue and propose a countermeasure.

If the initial verification is successful, the smart contract executes the second verification as outlined in lines 10-14. In this process, the fetched attributes are checked against the access policy $\mathcal{T}_i$ within the access structure. The access structure is defined by the patient during contract deployment and maintains the desired privacy settings as discussed in equations (4.1) and (4.2). The access policies are tested sequentially starting from the highest ranked access policy $\mathcal{T}_1$ at level $\mathcal{L}_1$. If the attributes satisfy the access policy $\mathcal{T}_i$, the verification is discontinued and the function fires an event LogAnnounce that announces a permanent log of the smart contract transaction stored over the blockchain. This event is a public and immutable announcement to ensure that the staff member in possession of signer should be granted access to the parts of the record $\{\mathsf{R}_i \ldots\ R_k\}$ corresponding to levels $\{\mathcal{L}_i \ldots \mathcal{L}_k\}$.

Granting keys: Granting Keys (GK) is a global smart contract that is used to share the location of the generated and encrypted access keys over the distributed storage. The contract generally consists of a single function, addKey, as outlined in Algorithm 3.

When the key-issuer observes a $\mathsf{LogAnnounce}$ event fired by the AVPA smart contract, it generates an access secret key SK for the specified staff member using the unique set of attributes $\mathsf{fetched}$ stored in the logs. Next, the key-issuer encrypts this access key as shown in Equation 4.3 with the public key of the staff member. It also encrypts the IPFS record location as $\mathsf{ERA} = \mathsf{Asym\text{-}Enc}_{\mathsf{Pub}}(\mathsf{\_ipfsRecordAddress})$ with the same public key and uploads both values to the IPFS storage, maintaining its reference location. Following that, the smart contract fires an event, $\mathsf{LogKeys}$ as shown in line 6, which permanently stores the address of the requesting staff member $\mathsf{\_staffAddress}$ along with the IPFS storage location $\mathsf{\_ipfsStorageAddress}$. Using $\mathsf{\_ipfsStorageAddress}$, the staff member can fetch the encrypted key ESK and ERA stored over the network. However, as shown in the addKey function, only certified key-issuers $\mathsf{setOfIssuers}$ are capable of triggering this function. Therefore, attackers are prevented from spamming the smart contract logs with fake keys or addresses. The staff member can listen for these fired events and obtain the corresponding $\mathsf{\_ipfsStorageAddress}$ as it appears in the logs. Using the $\mathsf{\_ipfsStorageAddress}$, the staff member can fetch ESK and ERA from the network. It is important to note that only the staff member in possession of the private key Pr corresponding to the public key Pub will be able to decrypt the fetched encrypted key and IPFS record address. Attackers continuously listening to the fired events

Algorithm 3 GK smart contract

```
1: event LogKeys(staffAddress, ipfsAddress)
2: function addKey(_staffAddress, _ipfsAddress)
3:     if (msg.sender $\notin$ setOfIssuers) then
4:         throw
5:     else
6:         LogKeys(_staffAddress, _ipfsStorageAddress)
7:     end if
8: end function
```

may be able to learn certain IPFS addresses storing generated encrypted keys, but not the actual keys or the record address. First the data user decrypts $\_\mathsf{ipfsRecordAddress} = \mathsf{Dec}_{\mathsf{Pr}}(\mathsf{ERA})$ to learn the location of the encrypted record. Next, using the obtained secret key SK, the staff member can then decrypt Esk_i to generate the secret key sk_i. Finally, the staff member can decrypt ER_i stored over IPFS at $\_\mathsf{ipfsRecordAddress}$ to obtain the record part R_i.

4.3.3 Access Permission Revocation

Attributes possessed by members may change over time due to, for example, job switch or retirement. As a result, the access rights to the patient records should be revoked to be consistent with the access policy. However, this could be challenging since it requires the attributes of the staff members to be updated periodically. While adding an expiration date [77] to each attribute when registering staff members seems to be a simple solution, it requires the patients to incorporate time constraints into their access policies.

The proposed d-MABE scheme can handle access permission revocation efficiently without requiring any extra process for the key-issuer. The staff members only need to be registered to ensure attribute-based access control. As a result, if at any point in time a staff member is unable to provide evidence of registration to the level of access claimed, future access to the patient records will be disabled. Consequently, if the staff member attempts to request records he/she is no longer entitled to access, the AVPA smart contract will fail to verify the request.

For data users that have already accessed certain records and are no longer registered, our proposed scheme can also revoke their access permissions since each record partition R_i is encrypted under $\mathsf{sk}_i \| \mathsf{SBK}$. To revoke access from those data users that are no longer registered, the patient only needs to generate a new subscription based key SBK then re-encrypt his/her record partitions under $\mathsf{sk}_i \| \mathsf{SBK}$. This design does not require regenerating a new set of keys $\{\mathsf{sk}'_1 \dots \mathsf{sk}'_k\}$ or making any changes to the privilege-based access structure. To prevent any data user that has been granted key sk_i at some point with an inactive membership from accessing the record partitions, we only need to re-encrypt the record partitions either periodically (such as monthly), or based on demand.

4.4 Security and Privacy Analysis

In this section, we present the security and privacy discussions of our proposed scheme. We assume our system runs over a blockchain that is maintained by a significant number of nodes, for example, Ethereum. In such blockchain platforms, it is very expensive to tamper with verified transactions processed into blocks. The best bet of the attackers would be to control more than half of the computational power in the network in order to perform 51% attacks. We also consider distributed storage networks such as IPFS that are content-addressable. These networks use the hash computation of the original data when storing and referencing it over the network nodes, resulting in tamper-resistant storage. Moreover, we consider adversaries that can act as any entity within the system and can manage to connect to either the IPFS or the blockchain network. Such adversaries are capable of generating fraudulent transactions and propagating them through the blockchain network. By fraudulent transactions, we mean transacting with smart contracts in an effort to access data to which they are not granted access. They may also deploy their own smart contracts over the blockchain, request/store data over the IPFS nodes, and continuously monitor the network channels.

4.4.1 Security Analysis

We first discuss how our proposed scheme is secure against potential attacks. Next, we present good practices that may help improve the security of the system.

Theorem 3. The proposed scheme is secure against replay attacks.

Proof. The initial verification process performed by the AVPA smart contract requires the requesting staff members to submit ECDSA digital signatures σ to prove their identities. Given that all data sent over the blockchain is public, malicious attackers can easily maintain a copy of the submitted signatures and later on impersonate the honest staff members giving them access to data they should not be able to access. To work around this issue, staff members are required to time-stamp their digital signatures before transacting with the AVPA contract, such that

$$\sigma \leftarrow \mathsf{Sign}(\mathsf{Pr}, \mathsf{Pub}, \mathsf{Hash}(\mathsf{staffID} \| \mathsf{T})), \tag{4.10}$$

where T is the time at which the signature is generated. Any submitted time-stamped signature is stored over the blockchain in order for the nodes to check if it has been submitted before. The blockchain nodes will not execute any requests associated with time-stamped signatures that have appeared previously. Therefore, to be able to perform replay attacks requires the attackers to reverse the transactions that contain an already used digital signature. The success of reversing a transaction can be modeled as a race between the honest miners and the attackers trying to generate blocks by competing to solve a hard cryptopuzzle known as Proof-of-Work [51]. The race can be modeled as a binomial random walk. It is denoted as z which represents the number of blocks generated by the honest miners with computational power p minus the number of blocks generated by the attackers with computational power $q = 1 - p$. This race can be derived as

$$z_{i+1} = \begin{cases} z_i + 1, & \text{with probability } p, \\ \\ z_i - 1, & \text{with probability } q, \end{cases} \tag{4.11}$$

Now, using the negative binomial distribution, we can model the probability of success of the attackers. Assume the key generator waits for n blocks to be generated by the honest miners with computational power p before generating a private key for the requesting attacker. Also assume that, at that time, the attacker is able to secretly generate m blocks with computational power $q = 1-p$, where $m = n-z-1$. By definition, this can be modeled as the m number of blocks that the attacker can generate before the n number of blocks generated by the honest miners. Therefore, the probability of a reversing a transaction for a given value m is calculated as

$$P(m) = \binom{m+n-1}{m} \times p^n q^m. \tag{4.12}$$

Overall, the probability for an attacker to surpass successfully the number of blocks generated by the honest miners can be computed as

$$
\begin{aligned}
P_s &= \sum_{m=0}^{\infty} P(m) \times Q_{n-m-1} \\
&= 1 - \sum_{m=0}^{\infty} \binom{m+n-1}{m} \times p^n q^m \times \begin{cases} 1 - \left(\frac{q}{p}\right)^{n-m}, & \text{if } q < p \text{ or } k \leq n-m, \\ 1 - 1 = 0, & \text{if } q > p \text{ or } k > n-m. \end{cases}
\end{aligned}
\tag{4.13}
$$

Equation (4.13) confirms that when $q > p$, the attacker will always succeed since $P_s = 1$. When $q < p$, the probability of success can be defined as

$$
\begin{aligned}
P_s &= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{k} \times p^n q^m \times \left(1 - \left(\frac{q}{p}\right)^{n-m}\right) \\
&= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{m} \times (p^n q^m - p^m q^n).
\end{aligned}
\tag{4.14}
$$

Therefore, as more blocks are appended to the blockchain and $q < p$, replay attacks are infeasible. □

Theorem 4. The proposed smart contracts are collusion resistant.

Proof. Our proposed scheme requires staff members to become registered through the SMR smart contract before being able to request access permissions through the AVPA contracts. During registration, each staff member address is mapped to a single set of attributes $\mathbb{A} = \{\mathsf{A}_1, \mathsf{A}_2 \ldots, \mathsf{A}_n\}$ after providing proof of possession to the certified institutions. The AVPA smart contract can only accept a single signature as input when triggered. Therefore, to perform a collusion attack, the attackers are required to regenerate a single digital signature of an already registered staff member that possesses the set of attributes desired. For ECDSA, a signature is generated by randomly selecting a cryptographically secure random integer d, such that

$$(x_1, y_1) = d \times G, \tag{4.15}$$

where (x_1, y_1) are the calculated elliptic curve points, G is the generator of the elliptic curve with a large prime order n, and $d \in_r [1, n-1]$. Next, the digital signature is calculated as two components $\sigma = (r, s)$. That is

$$r = x_1 \ mod \ n, \tag{4.16}$$

$$s = d^{-1}(z + r \cdot pr) \ mod \ n, \tag{4.17}$$

where z is the L_n leftmost bits of $\_\mathsf{msg}$ such that L_n is the bit length of the group order n. If any of the values r or s are equal to zero, d is randomly selected again and equations (4.15), (4.16), and (4.17) are recalculated. Since the signature components (r, s) are both derived based on the random value d, it is infeasible for the attackers to regenerate a specific signature that belongs to a certain staff member, hence, the smart contracts are collusion resistant. □

Theorem 5. Using a content-addressable storage such as IPFS ensures that data is tamper-resistant.

Proof. In IPFS, records are initially partitioned into n smaller segments before being stored. That is

$$\mathcal{R} = \{\mathsf{R}_1 \dots \mathsf{R}_n\}, \tag{4.18}$$

where each $\mathsf{R}_i \in \mathcal{R}$ is 256KB, by default. However, the size of a partition may also be customized by the patient as desired. Next, each $\mathsf{R}_i \in \mathcal{R}$ is cryptographically-hashed as

$$h_i = \mathsf{Hash}(\mathsf{R}_i), \tag{4.19}$$

where h_i is the resulting digest and is used to reference data stored in the network nodes through the DHT. For the Hash function, it is computationally infeasible to find an $\mathsf{R}_i' \neq \mathsf{R}_i$, such that

$$\mathsf{Hash}(\mathsf{R}_i') = \mathsf{Hash}(\mathsf{R}_i). \tag{4.20}$$

Therefore, it is computationally infeasible for an attacker to tamper with the records of the patients. □

Theorem 6. The proposed scheme can counter Distributed Denial of Service (DDoS) attempts.

Proof. DDoS attacks over IPFS aim at congesting the network by sending numerous requests to the nodes storing the requested data. However, to perform such attacks, the attackers must initially identify all the nodes storing the target data from the DHT. Next, they must disrupt the service by sending these nodes a large number of requests, such that

$$\text{Number of requests} \gg \mathsf{MAX\_TRAN}, \tag{4.21}$$

where $\mathsf{MAX\_TRAN}$ is the maximum number of transactions that are accepted by a node as defined in its configuration file.

Attackers may also try to isolate and monopolizes all inbound and outbound connections and data flows of the storage nodes (referred to as an Eclipse attack [78]) from the entire network. These attacks become infeasible as the number of nodes storing the data increases since the record is replicated across the network nodes. In addition, such attacks are limited by the time t_{scheme} for a request to appear over the blockchain until the staff member fetches the data from the IPFS network nodes. Attackers must be able to perform the entire attack in time t_{attack}. That is

$$t_{attack} < t_{scheme}. \tag{4.22}$$

An attacker with no prior knowledge of which data will be requested by staff members has no advantage of identifying the nodes storing the data. Therefore, as the value of t_{scheme} becomes smaller and with an increased record replication, DDoS attacks become infeasible. □

4.4.2 Recommended Security Practices

Security may also be enhanced by adopting good development practices. In the following, we present some highly recommended techniques that developers should keep in the back of their minds while implementing the system.

Global smart contracts access control: It is necessary to ensure the integrity of the data stored in the SMR and GK global contracts since they provide the necessary information required to verify AVPA contracts and grant keys to the staff members. Therefore, it is imperative to halt the attackers from manipulating the state of these contracts with fraudulent information. Attackers with access rights that can modify the SMR contract may easily register themselves with any set of attributes required to pass the verification of certain AVPA contracts. It is also feasible to perform DDoS attacks by deleting or modifying the registration of staff members in the SMR contract or spamming the logs of the GK contract with fraudulent information. This will result in interrupting or slowing down the process of retrieving patient records. Therefore, when developing such contracts, it is important to limit the ability to modify the state of global contracts to only certified institutions. This is outlined in lines 7 and 3 of Algorithms 1 and 3, respectively. The current smart contract programming languages, for example, Solidity [79], provide specific functions to verify certain conditions or variable values before execution is performed. Well-known examples include the require() or assert() functions that verify preliminary conditions and consume a portion or all of the gas associated with a transaction. In general, two main reasons for charging users gas to trigger smart contracts deployed over the blockchain are to reward the executing network nodes and make potential attacks expensive. This means that attackers must pay for each transaction they request for it to be executed by the network nodes. This approach will not prevent attackers from registering themselves into the SMR with a set of fraudulent attributes but could help halt those trying to spam the network. Assuming an attacker has been able to modify the state of a global smart contract, the attacker will still have to pay for each requested transaction. To increase the security measures in this scenario, smart contracts may be designed to require a minimum gas amount in order to execute any smart contract. However, this countermeasure results in a trade-off between security and cost for the honest users.

External smart contract calls: Our proposed scheme relies on external smart contract calls that fetch the stored attributes of registered staff members to compare them to those sent by the requesting staff members before validating if they satisfy a certain access policy. External calls can introduce several unexpected risks or errors if the smart contracts mistakenly execute malicious code. Therefore, it is important that patients cautiously implement their AVPA contracts to handle errors when externally calling the SMR global contracts during the initial verification process. In Solidity [79], external calls can be performed in two ways: low-level calls and contract calls. Low-level calls do not throw exceptions, instead they return false if the external call of another smart contract itself encounters an exception.

Patients that use low-level calls must check if the returned value within the AVPA contract fails and then properly handle it. On the other hand, contract calls are more direct as they throw exceptions as encountered by the external call. From a security standpoint, it may be more secure to use contract calls especially when the patient is less experienced in handling complex scenarios.

Non-Public Blockchains: The security of our proposed scheme could also be enhanced by running the smart contracts over a consortium or private blockchain. In such a setting, miners are selected nodes within the peer-to-peer network that are known and trusted. If any of these nodes attempt to act maliciously, they are immediately identified and eliminated. A good example of such candidate nodes could be the medical institutions themselves that are interested in maintaining the system in return for efficient data access when required.

4.4.3 Privacy Analysis

While our proposed scheme aims at satisfying the privacy desires of patients and their records, it tends to leak knowledge about the staff members and their access patterns. Each staff member must initially become registered and is linked to a unique address before engaging with the system. Since this information can be leaked, attackers can simply monitor requests triggered by the staff members by observing the blockchain activity. However, since no information is revealed about the records of patients through the AVPA smart contracts, attackers can only track when staff members trigger requests, but not the data they have requested or the patient information.

From the perspective of patients, our proposed scheme is intentionally designed to allow them to trace-back the accesses performed by staff members to their records. This is possible by tracing the history of LogAnnounce events fired as outlined in line 12 of Algorithm 2. All fired events are permanently stored over the blockchain allowing the patients to continuously monitor how their records are being accessed. If a patient notices irregular staff member accesses that are undesired, the patient can simply redeploy a new AVPA smart contract that defines new or more strict access policies.

Theorem 7. Under the proposed model, the privacy of patient records location is preserved.

Proof. First, since the user record is encrypted as described in equation (4.3) before it is uploaded to IPFS, the content of the user record is protected. Second, for the current IPFS system, any user in possession of the data can reproduce its cryptographic-hash, search its corresponding location that is maintained by the DHT, and locate it within the peer-to-peer network nodes determined by the default IPFS partitioning techniques. This will also result in recursively locating any data linked to it. To ensure data privacy, we will append a random value r to the record before uploading it to IPFS. Therefore, the storage location is determined by

$$h = \mathsf{Hash}(\mathcal{R} \| r). \tag{4.23}$$

The randomness of r makes it infeasible for the attacker to locate the data stored over IPFS nodes. □

4.5 Performance Analysis

In this section, we will evaluate the proposed smart contracts in terms of performance and monetary costs.

The performance measurement of smart contracts can be performed either through (i) calculating the gas costs required to deploy/transact with the contract, or (ii) measuring the computational complexity of the algorithms. Method (i) is usually more accurate since it presents an estimate of the actual monetary costs paid by the users in order to deploy/transact with the contract. However, this method is directly dependent on the actual source code implementation. That being said, developers that consider optimization techniques when coding their contracts can probably end up reducing the gas costs. In addition to this, even if we were to assume access to the entire source code of smart contracts, it is still difficult to determine the expected gas costs, since contracts may behave differently based on their state. On the other hand, method (ii) can help give us an indication of the computational complexity based on the number of inputs. This computational complexity directly correlates to the gas costs paid by the users. In other words, the more complex an algorithm is, the higher gas costs it will require to be deployed/executed. However, with smart contracts, the concept of reducing computational complexity to make it run faster is not of significant importance since blockchain transactions are generally executed at discrete intervals. Therefore, if we were to reduce the computational complexity, it would only be to reduce the gas costs required rather than the total latency.

To measure the performance of our proposed smart contracts, we will present discussions on each of our proposed algorithms that combine both methods. Our discussions assume that smart contracts will be implemented in Solidity [79], a contract-oriented, high-level language for implementing smart contracts deployable over the Ethereuem blockchain and processed by the EVM.

4.5.1 SMR Contract

As outlined in Algorithm 1, the addStaffMember function takes _attributes as input to register a new staff member. Due to the computational limitations of the EVM, returning dynamic content from external function calls is not possible, i.e. returning a dynamically sized array of attributes resulting from the getAttributes function when called by the AVPA smart contract. This means, the size of Map[_address].staffAttributes must be fixed in order to be executed by the EVM. As a result, the only workaround is to use statically-sized upBound arrays of attributes as input to the addStaffMember. Therefore, in this case, performance is mainly based on the size of upBound, such that

$$\text{Performance} \propto \mathsf{upBound}. \tag{4.24}$$

4.5.2 AVPA Contract

As discussed previously in Agorithm 2, the AVPA smart contract consists of two sequential verifications that play a dominant role in the performance of the contract. In the initial verification, an input digital signature σ is validated to prove the identity of the requesting staff member. With Solidity, the only available cryptographic function that can perform such a process is the ECDSA ecrecover function. The function recovers the Ethereum address associated with the public key from the elliptic curve signature or returns zero on error. At the time of writing, the ecrecover function requires 3000 gas units. In comparison to most of the available possible computations available by the EVM [34], the ecrecover function is considered to be relatively expensive. However, this initial verification is fixed with each AVPA contract transaction.

Assuming the initial verification is successful, the AVPA externally fetches the attributes of the staff member to test them against the incorporated access structure. Similar to the SMR contract, the performance of the AVPA contract is dependent on the upBound of the fetched attributes. Finally, the fetched attributes are tested against the access policies starting at the highest level within the hierarchy. Here, performance is affected by the complexity of the overall access structure defined and the number of levels k within the hierarchy. Assuming the worst case scenario, a requesting staff member might have his/her attributes tested against all access policies within the access structure and only receive the least sensitive part R_k of the record or nothing at all. The computational complexity can be represented as $\mathcal{O}(k \times \mathsf{upBound} \times |\mathcal{T}_i|)$, where $|\mathcal{T}_i|$ is the number of attributes that form the access policy at level $\mathcal{L}_i$. To reduce this complexity, we can modify the AVPA contract such that the staff members can request that their attributes be tested against only specific access policies rather than being tested sequentially against all policies. This would reduce the computational complexity to $\mathcal{O}(\mathsf{upBound} \times |\mathcal{T}_i|)$. We can also incorporate optimized search functions, for example, binary search, that can help optimize searching for attributes in the fetched set against those in the access policies. This means that we can further reduce the computational complexity to $\mathcal{O}(\mathsf{upBound} \times \log |\mathcal{T}_i|)$. However, it is important to note that in such a scenario, the patients need to make their access structures publicly available in order for the staff members to request certain access policies. This results in a trade-off between the performance of the AVPA contract and the privacy of the access policy defined.

4.5.3 GK Contract

The GK contract requires the least computational complexity and gas costs with respect to the SMR and AVPA smart contracts. The computations involved are as simple as firing a LogKeys event when access permissions are granted to staff members. The gas costs of such operations are minimal in comparison to the other computations in the SMR and AVPA smart contracts.

4.6 Empirical Results

In this section, we implement, simulate and present the estimated costs required to deploy/transact with the smart contracts of our proposed scheme in multiple scenarios. Our experiments are performed over the Ethereum testnet blockchain [80]. We specifically choose the Ethereum blockchain given the fact that it is by far the most widely used smart contract hosting blockchain. However, we note that our proposed scheme can also be implemented over any other blockchain that incorporates smart contracts.

In our simulation, the AVPA smart contract with the highest degree of complexity is implemented. When executed, the AVPA smart contracts sequentially check whether the attributes of the requesting users satisfy the access policies in order starting from the highest level of the hierarchy. As discussed in Section 4.5, there is a trade-off between privacy and performance. Therefore, we note that our presented results can be further enhanced in terms of performance as discussed previously. However, we intentionally choose to implement our smart contracts this way to show that even with these conditions, our proposed contracts are still cheaper when compared to the current systems used by the record-generating institutions to share medical records. In contrast to the current record-sharing systems, our proposed scheme eliminates the role of the record-generating institution completely. This results in eliminating other overhead costs required when sharing medical records. For example, in developed countries such as the U.S., record-generating institutions must comply with

Table 4.1: Estimated d-MABE smart contracts deployment at k=5 and N=25.

Smart Contract	$k = 5,\ N = 25$			
	Gas Limit	**Cost/ETH**	**Cost/USD**	**Data/bytes**
SMR	240383	0.004809	1.4	736
AVPA	857419	0.017148	5.21	3323
GK	125617	0.002512	0.74	304

Table 4.2: Estimated d-MABE smart contracts deployment at k=5 and N=50.

Smart Contract	$k = 5,\ N = 50$			
	Gas Limit	**Cost/ETH**	**Cost/USD**	**Data/bytes**
SMR	240447	0.004809	1.42	736
AVPA	1132628	0.022653	6.67	4536
GK	125617	0.002512	0.74	304

Table 4.3: Estimated d-MABE smart contracts deployment at k=10 and N=100.

Smart Contract	$k = 10,\ N = 100$			
	Gas Limit	**Cost/ETH**	**Cost/USD**	**Data/bytes**
SMR	240447	0.004809	1.42	736
AVPA	1303607	0.026072	7.56	5188
GK	125617	0.002512	0.74	304

certain state laws that enforce a maximum fee a record-generating institution may charge when requested to share its records. Table 4.4 illustrates a sample of these fees that could be inclusive or exclusive to the methods of delivering the record itself. Given the current competitiveness, in most cases, the medical institutions would charge the entire allowed fees to maximize their profits.

For our simulations, we apply the values $k = \{5, 10\}$, $\mathsf{upBound} = \{10, 20, 30, 40, 50\}$ and $N = \{25, 50, 100\}$, where k is the number of levels in the hierarchy and N is the total number of attributes used in the entire access structure. Without loss of generality, we also assume that the number of attributes for each level follows the normal distribution. For example, if $k = 5$ and $N = 50$, then the number of attributes used to define an access policy is $|\mathcal{T}_i| = 10$. The following experiments have been conducted through an Ethereum node running on a system with 1.8 GHz Intel Core i5 and 8 GB RAM. Our numerical results are

Table 4.4: U.S maximum state fees to copy and deliver records upon being requested by others.

State	Search fee	Cost per page	Misc. costs per page
California [81]	N/A	\$0.25	Microfilm: \$0.50
Texas [82]	\$82.95	P.1-10: \$45.79 flat fee P.11-60: \$1.54 P.61-400: \$0.76 P.401+: \$0.41	Microfilm: P.1-10: \$69.74 flat fee P.11+: \$1.54
Florida [83]	\$1.00/year	\$1.00	Microfilm: \$2.00
New York [84]	N/A	\$0.75	X-rays: Actual cost of reproduction
Illinois [85]	\$27.91	P.1-25: \$1.05 P.26-50: \$0.70 P.50+: \$0.35	Microfilm: \$1.74

Table 4.5: Estimated *d*-MABE addStaffMember function costs.

	upBound				
	10	**20**	**30**	**40**	**50**
Gas Limit	246531	449890	653249	856674	1060034
Cost/ETH	0.004931	0.008998	0.013065	0.017133	0.021201
Cost/USD	1.45	2.65	3.85	5.06	6.29

Table 4.6: Estimated *d*-MABE addKey function costs.

	upBound				
	10	**20**	**30**	**40**	**50**
Gas Limit	26379	26380	26375	26379	26378
Cost/ETH	0.000528	0.000528	0.00053	0.000524	0.000528
Cost/USD	0.14	0.14	0.14	0.15	0.15

also the averages of 10 trials under each scenario.

Based on our experiments, the costs of deploying our smart contracts are not affected by the value of upBound since there is no major change in code as this value changes. Therefore, for all values of upBound that we tested, the costs remained constant. Tables 4.1, 4.2, and 4.3 summarizes these costs in terms of gas limits and their estimated and equivalent costs in ETH and United States Dollar (USD) at the time of testing. To measure the optimum gas limit for each smart contract deployment, we used the JSON-RPC method, eth_estimateGas [86], that generates and returns an estimate of the required gas limit based on the network success rate. We also set the gas price to 20 GWEI, where 1 ETH $= 1 \times 10^9$ GWEI. We intentionally

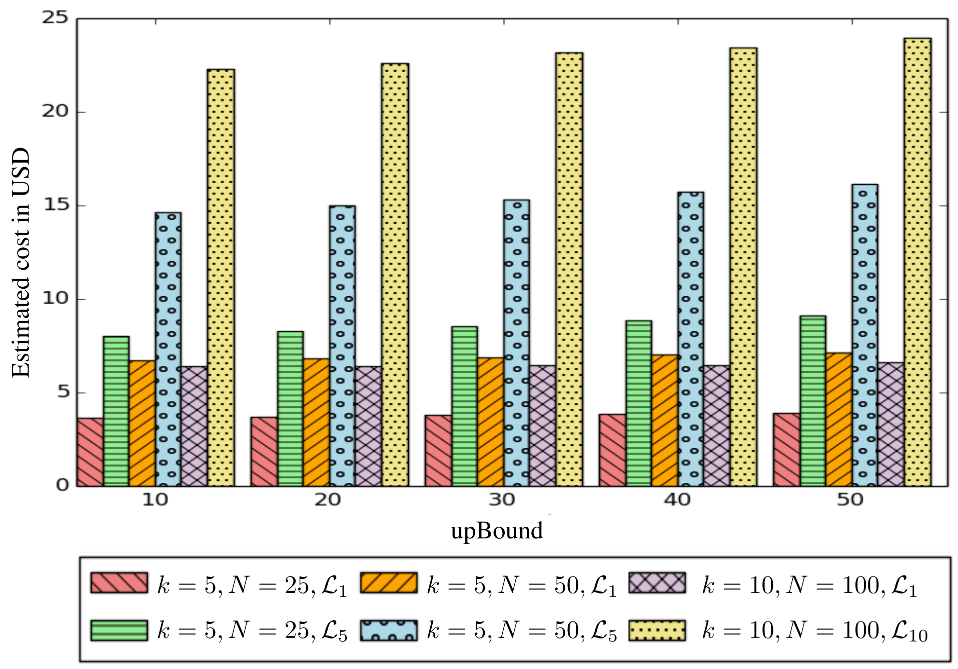


Figure 4.2: Estimated costs to run the d-MABE AVPA smart contract.

select this value relatively higher than the average gas prices specified in [87] for guaranteed and fast processing. Again, we note that our presented results can be further optimized by selecting reduced gas price values. As demonstrated in Tables 4.1, 4.2, and 4.3, the estimated costs for deploying the SMR and GK smart contracts remain constant as we alter the values of k and N. However, the costs for deploying the AVPA smart contracts increase with an increase in the values k or N.

Figure 4.2 demonstrates the changes in USD costs of transacting with already deployed AVPA smart contracts as we alter the values k and N for permissioned staff members at levels $\mathcal{L}_1$, $\mathcal{L}_5$ and $\mathcal{L}_{10}$. As shown by the figure, we recognize an increase in costs as these values increase. We also realize that as the upBound value increases while keeping the values k, N, and $\mathcal{L}_i$ constant, the costs slightly increase.

Finally, in Tables 4.5 and 4.6, we present the gas limits and costs in both ETH and USD to transact with the addStaffMember and addKey functions. As shown in Table 4.5, the costs of the transactions increase as the value upBound increases. On the contrary, as presented in Table 4.6, the costs of transacting with the addKey function remains constant regardless of the upBound value used.

4.7 Extended Application Discussions

Aggregated patient data has the capability of transforming the future standard of care for patients through the application of predictive analytics and big data methods. One of the biggest challenges to using health data to its fullest extent is the inaccessibility and segmentation of EMRs [88]. As demonstrated by our analyses, the proposed scheme can eliminate these constraints. It grants healthcare providers the ability to reliably and efficiently extract knowledge from disconnected EMR sources that have been willingly shared by patients.

For example, a meaningful opportunity for big data analytics in this context is the capability to model drug and treatment efficacy. Healthcare providers can access previously shared EMRs to understand how life-saving drugs and treatments perform, respective to the disease state and profile of the patient. This data can be used to make enhanced and customized treatment decisions for patients. As healthcare treatment becomes more individualized, successful patient outcomes are more likely and a one-size-fits-all approach to patient treatment is no longer adequate.

4.8 Summary

In this chapter, we proposed *d*-MABE, a secure, privacy-preserving and distributed data sharing scheme that runs over a blockchain using smart contracts. We demonstrated that *d*-MABE can be used in cases such as medical record-sharing where patients require an efficient and selective method to share their records with staff members of medical institutions. Our d-MABE proposed scheme eliminates reliance on the record-generating institutions when data is shared and completely empowers the patients. Our security and privacy analyses show that d-MABE is secure and preserves the privacy of patient records. We also presented some recommended security practices developers should keep in mind when developing their system. Finally, our comprehensive evaluation proves the efficiency of d-MABE and presents numerical results that support our performance analysis.

Chapter 5

d-CRAME: *Distributed* Coercion-Resistant and Anonymous Mobile Electronic Voting

5.1 Introduction

To further demonstrate the benefits of decentralization, we expand our research by proposing our second application. In this chapter, we introduce a novel voting scheme that enables free and fair large-scale elections. Our proposed scheme leverages the existence of at least two parties of an election with different allegiances that engage in a multi-party computation along with the voters. We assume that it conflicts with the interest of these parties to collude or exchange any information during the election process that may sacrifice the winning chances of the candidates they support. All computations can be performed remotely at the convenience of the parties involved. In addition, voters are able to cast their votes from a mobile device and verify whether their votes have been cast and counted properly. We design voter verifiability to be based on randomly generated values that even if shared with coercers willingly, will not provide any information on how the voters have voted. Furthermore, we utilize a blockchain that acts as a publicly accessible bulletin board that voters cast and store their votes to. We would like to make it clear that no computations of our proposed scheme are performed over the blockchain which will circumvent the scalability issues of blockchain in large-scale elections.

The rest of this chapter is organized as follows. Next, in Section 5.2, the problem formulation is described, outlining our design goals and system model. In Section 5.3, our proposed scheme is presented in detail outlining the proposed algorithms. Following that, in Section 5.4 we formally prove the security and privacy of our proposed scheme. In Section 5.5, a performance analysis and evaluation of *d*-CRAME are conducted and give our empirical results in Section 5.6. Finally, in Section 5.7, a conclusion is drawn to summarize the work done in this research.

5.2 Problem Formulation

Large-scale elections typically involve at least two parties with conflicting allegiances competing to win an election. Relying on a single entity to conduct a free and fair election between those parties requires significant trust in that entity to be unbiased. The trusted entity is responsible for multiple imperative tasks. Initially, it must properly register eligible voters prior to the voting phase. Next, it must authenticate voters during the voting process and provide them with a secure and coercion-resistant voting space for voters to cast their desired votes freely. Once the voting phase is over, the trusted entity must also fairly tabulate all votes, discarding the invalid ones and finally announce the winning candidates.

Given these constraints, large-scale elections are usually run or operated using in-person poll-sites. However, this may result in reduced voter turnout. While incorporating absentee ballots may improve voter turnout, it requires more trust not just in the organizing entity, but also during vote transmission. Even with stringent audits and monitoring, the entity performing any of those tasks may be able to cheat. Therefore, the challenge is to provide a complete voting process that all voters and running candidates can trust. This process must allow eligible voters to cast their votes remotely from anywhere while securing the integrity of the election and the safety of the voters.

5.2.1 Design Goals

Based on the problems described above, we have the following design goals:

Distributed trust: Organizing an election requires the involvement of at least two parties in which voters and candidates can trust. If a party attempts to cheat, it is exposed and disqualified from organizing future elections, incentivizing them to remain honest.

Voter eligibility: Voting is limited to eligible voters. This requires proper voter registration that determines and confirms the eligibility of voters, granting them voting rights.

Double-voting resistant: Each eligible voter is entitled to only a single vote counted toward the election. Submitting multiple votes would disqualify the previous votes, counting only the most recent one.

Anonymity: A cast vote cannot be linked to the identity of a voter. This protects voters, allowing them to freely voice their desired opinions.

Coercion-resistant: Remote voting may expose voters to coercion. If subject to coercion, voters can trick the coercers into believing that they have voted as they have been ordered, whilst still protecting their actual votes.

Voter verifiability: Voters can verify that their votes have been cast properly and counted toward the election results.

Election result manipulation-resistant: Last-minute voters cannot manipulate the election results in a close race. This requires concealing all votes until the voting phase is over.

5.2.2 System Model

Based on the design goals discussed, the system consists of a minimum of six entities, each with a distinct role.

Voters: The eligible set of voters $\{v_i \in \mathcal{V} \mid 1 \leq i \leq n\}$ that are granted the right to cast a vote in an election. This set is public and subject to audits to prove to the public that only eligible voters can vote.

Registrar $\mathcal{R}$: The first organizing entity, responsible for generating unique and random digital ballots to be shared with voters anonymously. It cannot link a digital ballot to its assigned voter.

Moderator $\mathcal{M}$: The second organizing entity, responsible for concealing the identities of voters and delivering the ballots to them anonymously. It cannot reveal the concealed digital ballots as it delivers them to the voters, hence, it cannot link a digital ballot to its assigned voter.

Election candidates: The eligible set of candidates $\{\mathsf{cand}_k \in \mathcal{C} \mid 1 \leq k \leq m\}$ running in an election.

Blockchain network: A non-trusted peer-to-peer network that maintains a publicly accessible blockchain and runs the election smart contract. The network nodes cannot link cast votes to voters or differentiate between valid and invalid votes.

Tallying authority: A party that performs vote tabulation at the end of the casting phase. This task is performed and monitored publicly and therefore does not require any trust.

5.3 The Proposed d-CRAME Scheme

Our proposed scheme consists of six main sequential phases, each occurring in a specific time frame predefined by the election organizer. We assume that the registrar $\mathcal{R}$ and the moderator $\mathcal{M}$ are two opposing parties in an election competing to win the election. Therefore, they are unlikely to collude. Let $\mathbb{G}$ be a publicly chosen multiplicative cyclic group of prime order p and g is a generator of $\mathbb{G}$.

5.3.1 Setup

The registrar and moderator each generate a key pair. They each select a secret key $x_r \in \mathbb{Z}_p^*$ and $x_m \in \mathbb{Z}_p^*$ randomly and then compute their corresponding public keys as $y_r = g^{x_r} \pmod{p}$ and $y_m = g^{x_m} \pmod{p}$, respectively.

5.3.2 Voter Registration

At this phase, voters are required to prove their voting eligibility to the registrar by providing evidence such as their identities. After validation, voters are added to the electoral roll.

```
Algorithm 4 Voter Registration
Input: Public key $y_i$ of $v_i$.
Output: Digital Signature $\mathsf{EG\text{-}Sign}_{y_r}(y_i)$.
Setup: (Registrar $\mathcal{R}$)
 1: Randomly select secret key $x_r \in \mathbb{Z}_p^*$.
 2: Compute public key as $y_r = g^{x_r} \pmod{p}$.
Registration Request: (Voter $v_i$)
 1: Randomly select secret key $x_i \in \mathbb{Z}_p^*$.
 2: Compute public key as $y_i = g^{x_i} \pmod{p}$.
 3: Send $y_i$ to $\mathcal{R}$.
Authentication: (Registrar $\mathcal{R}$) If $v_i$ is eligible:
 1: Randomly select $u_i$ with $1 < u_i < p-1$ and $\gcd(u_i, p-1) = 1$.
 2: Compute $w_i = g^{u_i} \pmod{p}$.
 3: Compute $s_i = (h(y_i) - x_r w_i)u_i^{-1} \pmod{p}$.
 4: Send $(w_i, s_i)$ to $v_i$.
```

Algorithm 4 summaries this process.

5.3.2.1 Voter Key Generation and Registration

Each voter v_i selects a secret key $x_i \in \mathbb{Z}_p^*$ randomly and then computes the corresponding public key as $y_i = g^{x_i} \pmod{p}$. Public keys are shared with the registrar to complete registration.

5.3.2.2 Signing Voter's Public Key

The registrar verifies the eligibility of the voters and signs their public keys to add them to the electoral roll. It selects u_i randomly where $1 < u_i < p-1$ and $\gcd(u_i, p-1) = 1$ then computes the following:

$$w_i = g^{u_i} \pmod{p}, \tag{5.1}$$

$$s_i = (h(y_i) - x_r w_i)u_i^{-1} \pmod{p}, \tag{5.2}$$

where (w_i, s_i) is the signature. The registrar discloses the electoral roll of public keys once registration is complete.

5.3.3 Acquiring a Ballot

To cast a vote, each voter must acquire a digital ballot. Algorithm 5 summarizes this process.

5.3.3.1 Ballots Generation

The registrar generates n unique and random digital ballots $\mathcal{T} = \{t_i \mid i \in \mathbb{Z}_n\}$ then digitally signs them using ElGamal signature scheme. We denote the set as $\mathcal{B} = \{\mathsf{bal}_i = \mathsf{EG\text{-}Sign}(t_i) \mid i \in \mathbb{Z}_n\}$. Next, it performs a permutation π on $\mathcal{B}$ such that:

$$\pi \colon \mathcal{B} \to \mathcal{B}.$$

5.3.3.2 Voter Permutation

The moderator is an intermediary that conceals voter identities from the registrar during ballot distribution. It generates a one time permuted set σ of n unique numbers used with every voter ballot request such that:

$$\sigma \colon \mathbb{Z}_n \to \mathbb{Z}_n.$$

5.3.3.3 Requesting a Ballot

Voters initiate the request by sharing their signed public keys with the moderator.

```
Algorithm 5 Ballot Acquisition
Input: Public key $y_i$ and it's signature $\mathsf{EG\text{-}Sign}_{y_r}(y_i)$ of $v_i$.
Output: Ballot $\mathsf{bal}_i$
Ballot Generation (Registrar $\mathcal{R}$):
  1: Generate random ballots as $\mathcal{T} = \{t_i \mid i \in \mathbb{Z}_n\}$.
  2: Digitally sign ballots as $\mathcal{B} = \{\mathsf{bal}_i = \mathsf{EG\text{-}Sign}(t_i) \mid i \in \mathbb{Z}_n\}$.
  3: Perform permutation $\pi\colon \mathcal{B} \to \mathcal{B}$.
Voter Permutation (Moderator $\mathcal{M}$):
  1: Generate permutation as $\sigma\colon \mathbb{Z}_n \to \mathbb{Z}_n$.
Request Ballot (Voter $v_i$):
  1: Send public key $y_i$ and $\mathsf{EG\text{-}Sign}_{y_r}(y_i)$ to the moderator.
Voter Key Validation (Moderator $\mathcal{M}$):
  1: Check $0 < w_i < p$ and $0 < s_i < p - 1$.
  2: Verifies $g^{h(y_i)} \equiv y_i^{w_i}, w_i^{s_i} \pmod{p}$. If Valid, $\Gamma = \text{true}$; otherwise $\Gamma = \text{false}$.
If $\Gamma = \text{true}$:
Voter Key Obscuring (Moderator $\mathcal{M}$):
  1: Randomly select $b_i \in \mathbb{Z}_p^*$.
  2: Compute $y_i' = y_i^{b_i} = g^{x_i b_i} \pmod{p}$.
  3: Send $y_i'$ to $\mathcal{R}$.
Ballot Assignment and Encryption (Registrar $\mathcal{R}$):
  1: Select ballot as $\mathsf{bal}_i = \pi(\sigma(i))$.
  2: Compute key as $k_i = (y_i')^{q_i} = g^{x_i b_i q_i} \pmod{p}$.
  3: Encrypt ballot as $\mathsf{ebal}_i = \mathsf{AES\text{-}Enc}_{k_i}(\mathsf{bal}_i)$.
  4: Compute ephemeral key as $Q_i = g^{q_i} \pmod{p}$.
  5: Send $\mathsf{ebal}_i$ and $Q_i$ to $\mathcal{M}$.
Ballot Transmission (Moderator $\mathcal{M}$):
  1: Encrypt $eb_i = (g^{rm}, b_i \cdot y_i^{rm}) = (c_1, c_2)$
  2: Send $\mathsf{ebal}_i$, $Q_i$, and $eb_i$ to $v_i$.
Derive Ballot (Voter $v_i$):
  1: Decrypt $b_i = c_2 \cdot c_1^{-x_i} = b_i \cdot y_i^{rm} \cdot (g^{rm})^{-x_i}$.
  2: Compute $k_i = (Q_i)^{x_i b_i} = g^{q_i x_i b_i} \pmod{p}$.
  3: Decrypt $\mathsf{bal}_i = \mathsf{AES\text{-}Dec}_{k_i}(\mathsf{ebal}_i)$.
```

5.3.3.4 Voter Identity Obscuring

The moderator compares the public key with the electoral roll list to verify eligibility. It validates the signature (w_i, s_i) corresponding to the shared public key y_i by verifying whether the following equation:

$$g^{h(y_i)} \equiv y_i^{w_i}, w_i^{s_i} \pmod{p}, \tag{5.3}$$

holds true. In this event, the moderator obscures y_i by selecting a blind factor $b_i \in \mathbb{Z}_p^*$ randomly and computing the following:

$$y_i' = y_i^{b_i} = g^{x_i b_i} \pmod{p}. \tag{5.4}$$

The moderator then sends y_i' and $\sigma(i)$ to the registrar.

5.3.3.5 Ballot Assignment and Encryption

Ballots are assigned randomly by the registrar to the blinded voters as follows:

$$\mathsf{bal}_i = \pi(\sigma(i)). \tag{5.5}$$

To conceal bal_i, the registrar encrypts it under an encryption key k_i derived as:

$$k_i = (y_i')^{q_i} = g^{x_i b_i q_i} \pmod{p}, \tag{5.6}$$

where $q_i \in \mathbb{Z}_p^*$ is selected randomly. Using k_i, the registrar encrypts the ballot as the following:

$$\mathsf{ebal}_i = \mathsf{AES\text{-}Enc}_{k_i}(\mathsf{bal}_i), \tag{5.7}$$

where $\mathsf{AES\text{-}Enc}$ is the AES encryption function. The purpose of this encryption is to conceal the ballot from the moderator. It enables the registrar to share this ballot with the voter anonymously. For this purpose, it generates an ephemeral key Q_i that would allow the voter to regenerate k_i such that:

$$Q_i = g^{q_i} \pmod{p}. \tag{5.8}$$

Finally, the registrar sends ebal_i and Q_i to the moderator.

5.3.3.6 Encrypted Ballot Transmission

Once received, the moderator sends ebal_i and Q_i to the voter along with the encryption of the blind factor b_i as:

$$eb_i = (g^{rm}, b_i \cdot y_i^{rm}) = (c_1, c_2), \tag{5.9}$$

where $r_m \in \mathbb{Z}_p^*$ is selected randomly.

5.3.3.7 Deriving Ballot

The voter can decrypt eb_i and recover b_i as:

$$b_i = c_2 \cdot c_1^{-x_i} = b_i \cdot y_i^{rm} \cdot (g^{rm})^{-x_i}. \tag{5.10}$$

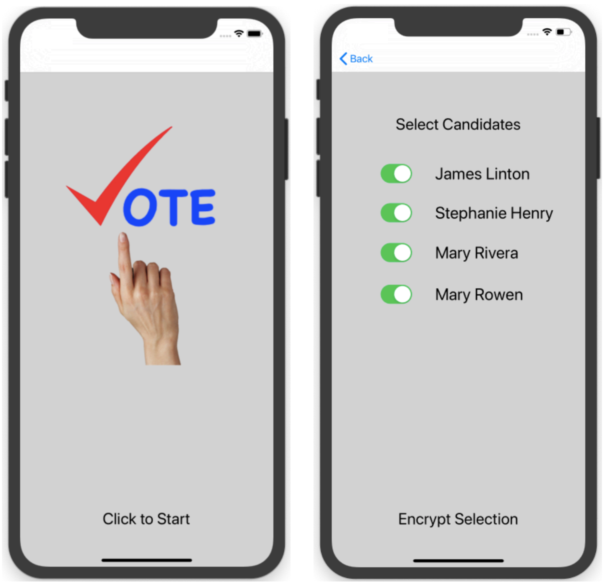
Next, the voter regenerates key k_i as:

$$k_i = (Q_i)^{x_i b_i} = g^{q_i x_i b_i} \pmod{p}. \tag{5.11}$$

Finally, the voter decrypts ebal_i as:

$$\mathsf{bal}_i = \mathsf{AES\text{-}Dec}_{k_i}(\mathsf{ebal}_i), \tag{5.12}$$

where $\mathsf{AES\text{-}Dec}$ is the AES decryption function.



(1) (2)

Figure 5.1: *d*-CRAME steps to select the desired candidates: (1) application start and (2) selecting and encrypting selected candidates.

5.3.4 Casting Votes

Before casting a vote, voters select the desired candidates they wish to vote for. Fig. 5.1 represents our designed mobile application showing candidate selection. Next, the voter proceeds with encrypting and casting the vote.

5.3.4.1 Ballot Double-Encryption

The ballot associated with a vote, denoted as B_i, is encrypted under the public keys y_r and y_m of both the registrar and moderator as:

$$B_i = (g^v, T \cdot (y_r \cdot y_m)^v) = (c_3, c_4), \tag{5.13}$$

where $v \in \mathbb{Z}_p^*$ is selected randomly, $T = (\mathsf{bal}_i \parallel \text{Vote})$, and $\text{Vote} = (\mathsf{cand}_1, \mathsf{cand}_2, \ldots, \mathsf{cand}_m)$ is a sequence of bits representing each candidate such that:

$$\mathsf{cand}_k = \begin{cases} 1, & \text{if voting for } \mathsf{cand}_k, \\ 0, & \text{if voting against } \mathsf{cand}_k. \end{cases} \tag{5.14}$$

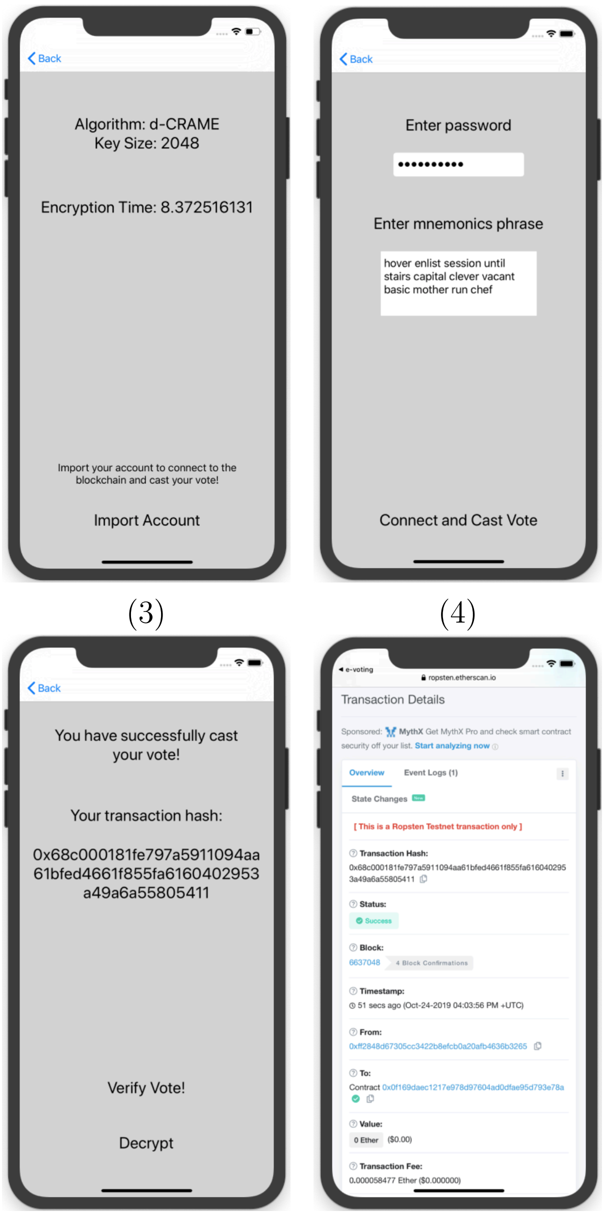
5.3.4.2 Submit Vote

To submit the encrypted ballot, the voter calls the election vote smart contract $\mathsf{SC}_{\mathsf{vote}}$ and integrates B_i as input. A vote is permanently cast once the result of the smart contract is appended to the blockchain. Fig. 5.2 is a summary of the steps performed by the voter to cast the encrypted vote. This process involves importing the blockchain account of the voter that allows her to call the casting smart contract deployed over the blockchain. Once the vote is cast, the voter receives a confirmation of the vote along with the transaction hash reference. The transaction hash is used to verify that the cast vote is stored permanently over the blockchain.

5.3.5 Tabulation

Votes that appear on the blockchain within the voting phase are collected to be validated and counted. Both the moderator and registrar publicly disclose their secret keys x_m and x_r. The registrar also discloses a new permutation $\gamma\colon \mathcal{B} \to \mathcal{B}$ containing all ballots distributed among voters during the acquiring ballots phase. The tallying authority decrypts the attached ballots appearing on the blockchain as:

$$T = c_4 \cdot c_3^{-x_m} \cdot c_3^{-x_r} = T \cdot (y_r \cdot y_m)^v \cdot (g^v)^{-x_m} \cdot (g^v)^{-x_r}. \tag{5.15}$$


Back

Algorithm: d-CRAME
Key Size: 2048

Encryption Time: 8.372516131

Import your account to connect to the blockchain and cast your vote!

Import Account

(3)

Back

Enter password

••••••••••

Enter mnemonics phrase

hover enlist session until stairs capital clever vacant basic mother run chef

Connect and Cast Vote

(4)

Back

You have successfully cast your vote!

Your transaction hash:

0x68c000181fe797a5911094aa61bfed4661f855fa6160402953a49a6a55805411

Verify Vote!

Decrypt

(5)

◂ e-voting
ropsten.etherscan.io

Transaction Details

Sponsored: MythX Get MythX Pro and check smart contract security off your list. Start analyzing now

Overview Event Logs (1)

State Changes New

[This is a Ropsten Testnet transaction only]

Transaction Hash:
0x68c000181fe797a5911094aa61bfed4661f855fa6160402953a49a6a55805411

Status:
Success

Block:
6637048 4 Block Confirmations

Timestamp:
51 secs ago (Oct-24-2019 04:03:56 PM +UTC)

From:
0xff2848d67305cc3422b8efcb0a20afb4636b3265

To:
Contract 0x0f169daec1217e978d97604ad0dfae95d793e78a

Value:
0 Ether ($0.00)

Transaction Fee:
0.000058477 Ether ($0.000000)

(6)

Figure 5.2: *d*-CRAME steps to cast and verify vote: (3) encryption result, (4) importing blockchain account to cast vote, (5) information on block containing cast vote, and (6) verifying vote permanently exists on blockchain.

If $\mathsf{bal}_i \in \gamma$, the tallying authority examines the attached vote sequence and increments the counter of each candidate accordingly. Each election may specify its own rules that disqualify votes which are cast incorrectly. For example, voting *yes* for two opposing candidates.

5.3.6 Vote Verifiability

At the end of the tabulation phase, the tallying authority publishes the results of the election on the blockchain. Each vote is published with its corresponding ballot as proof of the legitimacy of the results. Voters can simply recognize their ballots and verify that their votes have been counted properly toward the election. The moderator also verifies that only eligible voters have submitted votes since it has the list of registered voters. This prevents the registrar from attempting to issue unassigned ballots to voters of its own if they have not been claimed by registered voters.

5.4 Security and Privacy Analysis

In this section, a formal proof of security and privacy is presented.

Unlike the conventional elections where eligible voters are granted the right to voice their opinion exactly one time, there are many possible new security and privacy issues for electronic and remote voting. The malicious act of attempting to cast more than one vote is referred to as *double-voting* and aims to give an election candidate an advantage in winning the election over others. In countries such as the United States, while the majority of states prohibit voting twice in the same election, only a few of them prohibit voting in more than one state [89]. This means that eligible voters could register in more than one state and attempt to double-vote. The process of detecting and penalizing such voters becomes expensive and challenging. In remote electronic-voting systems, sometimes referred to as internet-voting (i-voting), voters receive digital ballots and cast their votes remotely rather than visiting a polling station. Given this flexibility, elections running over i-voting schemes may even allow voters to cast their votes multiple times using the same digital ballot, counting only their final vote and discarding all previous ones. Since our proposed scheme uses the blockchain as a bulletin board to permanently store votes, it becomes simple to identify double-voting attempts that reuse digital ballots. However, the reuse of digital ballots is not the only method to attempt double-voting. Adversaries may attempt to double-vote by obtaining undeserved voting credentials giving them the right to cast more votes. We formally prove that an election running our proposed scheme is secure against such attempts.

Definition 2 (Secure against double-voting)**.** A voting scheme is said to be secure against double-voting if no PPT adversary is able to forge a digital ballot that is digitally signed by the registrar.

Theorem 8. It is infeasible for any adversary to generate a legitimate ballot that can be used to cast a vote correctly if the DDH assumption holds.

Proof. Each ballot $\mathsf{bal}_i \in \mathcal{T}$ is digitally signed by the registrar before being distributed among the anonymous voters. For the adversary to generate acceptable ballots, it must be able to forge a signature of a ballot $\mathsf{bal}'_i = \mathsf{EG\text{-}Sign}(t'_i)$. This requires the adversary to learn the secret key x_r of the registrar or find collisions such that $h(\mathsf{bal}'_i) = h(\mathsf{bal}_i)$. Both operations can be reduced to the discrete logarithm problem. Therefore, it is infeasible for the adversary to generate a signed ballot correctly and the proof is complete. $\square$

To preserve the anonymity of voters, an adversary should not be able to link any vote to a specific voter. The proposed scheme relies on a secure multi-party computation performed by parties of different allegiances to address this issue. It requires a minimum of two conflicting parties to participate during the ballot distribution process. As demonstrated, a moderator conceals the public key y_i of the voter using a blind factor f_i and associates it with a random value selected from the permutation $\sigma(i)$. On the other hand, the registrar selects a ballot randomly and assigns it to the anonymous voter. The moderator and registrar would need to collude for ballots to be linked to the identities of voters. We assume collusion is not in the best interest of any of these parties, therefore, voter anonymity can be preserved. Our proposed scheme is considered to be secure under this assumption that the probability of an adversary to identify a public key y_i that has been randomly selected from a two-element public key chosen by the adversary and encrypted does not significantly exceed $\frac{1}{2}$. Using the Indistinguishibility under Chosen-Plaintext Attack (IND-CPA) Security Game, we provide a formal proof of security for our proposed scheme. We denote the DDH oracle as $\mathcal{O}_1 = (\mathsf{BFGen}, \mathsf{Blind}, \mathsf{Rec})$, where BFGen is the blind factor generation function, Blind is the blind function, and Rec is the recovery function. The IND-CPA game consists of a set of interactions between two PPT machines, an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ acting as the moderator.

1. $\mathcal{C}$ computes a blind factor $f = \mathsf{BFGen}(1^k)$ and keeps it secret.

2. Since $\mathcal{A}$ does not have access to $\mathcal{O}_1$, it may request that $\mathcal{C}$ blinds for it as many public addresses as it likes during any time of the game. $\mathcal{A}$ then computes two public addresses y_0 and y_1 to be challenged against and sends them to $\mathcal{C}$.

3. $\mathcal{C}$ uniformly and randomly selects $\mu \in_r \{0, 1\}$ then computes $y' = \mathsf{Blind}(f, s, y_\mu)$, where s represents a randomness state to diversify the blind process and is a value that has not been used in any of the previously computed ciphertexts. Next, $\mathcal{C}$ sends y' to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a guess μ' of μ. $\mathcal{A}$ wins the security game if $\mu' = \mu$ and loses otherwise.

An adversary that can derive which public key was blinded in polynomial time may be able to identify the identities of the voters and link them to the ballots being assigned. The

DDH assumption implies that the adversary is unable to get a non-negligible advantage from the IND-CPA Security Game in determining the public key that is blinded.

Definition 3 (Voter anonymity)**.** A voting scheme is said to preserve the anonymity of voters if no PPT adversary is able to get a non-negligible advantage by performing the IND-CPA Security Game protocol, i.e. $Adv = Pr[\mu' = \mu] < \frac{1}{2} + \varepsilon$ for any given negligible ε.

Theorem 9. The proposed scheme preserves the anonymity of voters if the DDH assumption holds.

Proof. Assume there is an adversary that has non-negligible advantage ε, then $\mathsf{Adv}_{\mathcal{A}} > \frac{1}{2} + \varepsilon$. We construct a simulator $\mathcal{A}$ that can distinguish a DDH element from a random element with advantage ε. Let $\mathbb{G}$ be a publicly chosen multiplicative cyclic group of prime order p. The DDH challenger begins by selecting the random parameters: $a, b \in_r \mathbb{Z}_p^*$. Let $g \in \mathbb{G}$ be a generator and T is defined as $T = g^{ab} \pmod p$ if $\mu = 0$, and $T = g^c \pmod p$ for some random $c \in \mathbb{Z}_p^*$ otherwise, where $\mu \in_r \{0, 1\}$. The simulator acts as the challenger in the IND-CPA game.

1. $\mathcal{C}$ chooses a blind factor $f \in_r \mathbb{Z}_p^*$ and state $s^* \in_r \mathbb{Z}_p^*$ then computes $s = s^* + ab$ and keeps them secret.

2. $\mathcal{A}$ chooses two secret keys $x_0, x_1 \in_r \mathbb{Z}_p^*$ then computes their corresponding public addresses y_0 and y_1 and sends them to $\mathcal{C}$.

3. $\mathcal{C}$ uniformly and randomly selects $\mu \in_r \{0, 1\}$ then computes $y^* = \mathsf{Blind}(f, s, y_\mu) = y_\mu^f g^s = g^{x_\mu f} g^{s^*+ab} = g^{x_\mu f} g^{s^*} T \pmod p$, where $T = g^{ab} \pmod p$. Next, $\mathcal{C}$ sends y^* to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a guess μ' of μ. $\mathcal{A}$ wins the security game if $\mu' = \mu$ and loses otherwise.

Given $\mathcal{A}$, if $T = g^{ab} \pmod p$, then y^* is a valid ciphertext, $Adv = \varepsilon$ and

$$\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{ab}\right) = 1\right] = \frac{1}{2} + \varepsilon. \tag{5.16}$$

If $T = g^c \pmod p$ *or* $T \neq g^{ab} \pmod p$ then y^* is nothing more than a random value to the adversary. Therefore,

$$\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{c}\right) = 1\right] = \frac{1}{2}. \tag{5.17}$$

From equation (5.16) and equation (5.17), we can conclude that

$$\left|\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{ab}\right) = 1\right] - \mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{c}\right) = 1\right]\right| = \varepsilon. \tag{5.18}$$

The simulator plays the DDH game with a non-negligible advantage which contradicts the DDH assumption. Therefore, the adversary cannot have advantage ε and the proof is complete. $\square$

In traditional paper ballot based voting systems, voters can cast their votes in a physically isolated environment without facing interference from adversaries throughout the election process. In i-voting systems, voting is performed remotely by the voters, therefore, it is possible that an adversary may be able to monitor the vote casting process. To address this concern, a stronger form of private voting known as coercion-resistance emerged [90]. A coercion-resistant voting system is one that accounts for an adversary that can engage with voters while they cast their votes during an election. Engagement may be in the form of an adversary coercing voters to cast their votes in a specific form or even forcing them to divulge their voting credentials by easily blackmailing the voters. Other engagements may even include an adversary willing to peacefully buy votes from the voters. Therefore, a practical coercion-resistant voting system should be *receipt-free*, allowing voters to evade proving to their coercers how they voted. This property requires a voting system to be designed in a way so that it does not generate any legitimate evidence that may leak information about the votes. Concurrently, the voting system should still allow users to *recognize* their votes individually and check whether their votes are being counted toward the final election results properly. In comparison to the receipt-free property, vote recognition is based on randomness, providing voters with values that correspond to their votes to give them the assurance needed to prove that their votes have been counted correctly. Even if shared by the voters with their coercers willingly, these values should not suffice to expose any vote information.

In addition to individual verifiability, e-voting systems could be designed to provide universal verifiability. This feature is intended to allow all voters and any auditors to verify the legitimacy of each cast vote and validate the integrity of the election results [91]. However, by definition, this property is orthogonal to coercion-resistance [92]. That is, a system that would allow any individual to verify every vote may also leak information about how voters cast their votes. From a practical and feasible standpoint, we believe that the coercion-resistance property is more significant for practical i-voting schemes. Not only does it allow voters to freely cast their votes as they desire while defrauding any type of adversary, but it also provides each voter with individual verifiability. Therefore, our goal is to introduce a coercion-resistance i-voting scheme where we assume the adversaries can interfere with the election but it is infeasible for them to distinguish between any real voting credentials and defrauded ones they may obtain from the coerced voters.

In the proposed scheme, deriving valid credentials used to cast a vote traces back to the voters acquiring the correct blind factors used to keep their identities anonymous as shown in equation (5.10). To prove that the proposed scheme is coercion-resistant, we present the following Indistinguishability of Encryption Keys (IND-EK) Security Game consisting of a DDH oracle $\mathcal{O}_2 = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is a public key pair generator, Enc is the encryption function, and Dec is the decryption function. The IND-EK game is a set of interactions between two PPT machines, an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

1. $\mathcal{C}$ computes two pairs of keys $\mathsf{KeyGen}(1^k) \rightarrow (y_0, x_0)$ and $\mathsf{KeyGen}(1^k) \rightarrow (y_1, x_1)$ then sends the public keys y_0 and y_1 to $\mathcal{A}$ while keeping x_0 and x_1 secret.

2. $\mathcal{A}$ has access to $\mathcal{O}_2$ and can encrypt as many blind factors of its choice as it chooses with any of the keys it receives. Next, $\mathcal{A}$ chooses a blind factor f and sends it to $\mathcal{C}$.

3. $\mathcal{C}$ uniformly and randomly selects $\mu \in_r \{0, 1\}$ then computes $c^* = \mathsf{Enc}(y_\mu, f)$. Next, $\mathcal{C}$ sends c^* to $\mathcal{A}$.

4. $\mathcal{A}$ outputs a guess μ' of μ. $\mathcal{A}$ wins the security game if $\mu' = \mu$ and loses otherwise.

An adversary that can derive which public key was used in encryption efficiently may potentially be able to link the votes to the actual voters. As a result, the adversary may be able to discover whether the coerced voter has behaved as instructed. The DDH assumption implies that the adversary is unable to get a non-negligible advantage from the IND-EK Security Game in determining the public key of the corresponding private key.

Definition 4 (Coercion-resistance)**.** A voting scheme is said to be coercion-resistant if no PPT adversary is able to get a non-negligible advantage by performing the IND-EK Security Game, i.e. $\mathsf{Adv}_{\mathcal{A}} = Pr[\mu' = \mu_i] < \frac{1}{2} + \varepsilon$ for any negligible ε.

Theorem 10. The proposed scheme is coercion-resistant if the DDH assumption holds.

Proof. Assume there is an adversary that has non-negligible advantage ε, i.e., $\mathsf{Adv}_{\mathcal{A}} > \frac{1}{2}+\varepsilon$. We construct a simulator $\mathcal{A}$ that can distinguish the public key used to encrypt a blind factor with ε. Let $\mathbb{G}$ be a publicly chosen multiplicative cyclic group of prime order p. The DDH challenger begins by selecting the random parameters: $a, b \in_r \mathbb{Z}_p^*$. Let $g \in \mathbb{G}$ be a generator and T is defined as $T = g^{ab} \pmod p$ if $\mu = 0$, and $T = g^c \pmod p$ for some random $c \in \mathbb{Z}_p^*$ otherwise, where $\mu \in_r \{0, 1\}$. The simulator acts as the challenger in the following game:

1. $\mathcal{C}$ chooses the parameters $x_0^*, x_1^* \in_r \mathbb{Z}_p^*$ then computes $x_0 = x_0^* + a$ and $x_1 = x_1^* + a$. Next, it simulates $y_0 = g^{x_0} \leftarrow g^{x_0^*+a} \pmod p$ and $y_1 = g^{x_1} \leftarrow g^{x_1^*+a} \pmod p$. Finally, it sends y_0 and y_1 to $\mathcal{A}$ and keeps x_0 and x_1 secret.

2. $\mathcal{A}$ chooses a blind factor $f \in \mathbb{Z}_p^*$ and sends it to the simulator.

3. $\mathcal{C}$ uniformly and randomly selects $\mu \in_r \{0, 1\}$ then computes $c^* = \mathsf{Enc}(y_\mu, f) = (g^b, fy_\mu^b) = (g^b, fg^{x_\mu b}) = (g^b, fg^{(x_\mu^*+a)b}) = (g^b, fTg^{x_\mu^*}) \pmod p$, where $T = g^{ab} \pmod p$. Next, $\mathcal{C}$ sends c^* to $\mathcal{A}$.

4. If $\mathcal{A}$ guesses the correct value, the challenger outputs 0 to indicate that $T = g^{ab}$, or 1 to indicate that $T = R$, a random group element in $\mathbb{G}$.

Given $\mathcal{A}$, if $T = g^{ab} \pmod p$, then c^* is a valid ciphertext, $Adv = \varepsilon$ and

$$\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{ab}\right) = 1\right] = \frac{1}{2} + \varepsilon. \tag{5.19}$$

$T = g^c \pmod p$ *or* $T \neq g^{ab} \pmod p$ then y^* then c^* is nothing more than a random value to the adversary. Therefore,

$$\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^c\right) = 1\right] = \frac{1}{2}. \tag{5.20}$$

From equation (5.19) and equation (5.20), we can conclude that

$$\left|\mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^{ab}\right) = 1\right] - \mathsf{Pr}\left[\mathcal{A}\left(g, g^a, g^b, T = g^c\right) = 1\right]\right| = \varepsilon. \tag{5.21}$$

The simulator plays the DDH game with a non-negligible advantage which contradicts the DDH assumption. Therefore the adversary cannot have advantage ε and the proof is complete. □

The proposed scheme utilizes a blockchain as its public bulletin board for voters to cast their votes. Voters interact with a voting smart contract that accepts a vote in the form shown by equation (5.13) as input. At the end of the voting phase, the tallying authorities scan the blockchain logs to collect all votes that have been cast during the voting phase. Votes that pass the validation are counted towards the election results while those that fail are discarded. Valid votes are posted to the blockchain along with their corresponding ballots to announce election results and allow voter validation. Voters can individually recognize that their votes have been counted correctly towards the final election results.

Before casting their votes to the blockchain, voters are required to encrypt their votes under the public keys x_r and x_m of the registrar and moderator. The encryption conceals the actual vote during the voting phase and protects the integrity of the results from last-minute voters that may try to manipulate the election results in a close race by voting in favor of a certain candidate or party. However, encryption alone may be insufficient to prevent an adversary from attempting to manipulate the election results. An adversary with significant computational power may attempt to remove votes that have already been cast from the blockchain to reduce the count of a candidate or party. To do this, the adversary must compete in the blockchain mining process to fork the blockchain and remove certain blocks carrying specific votes.

Theorem 11. The proposed scheme is secure against election result manipulation if the computational power of the adversary in the blockchain network is $q < 1/2 < p$ where p is the computational power of the honest miners.

Proof. The race of generating a block can be modeled as a binomial random walk. Let z be the number of blocks generated by the honest miners minus the number of blocks generated by the adversary. This race can be derived as:

$$z_{i+1} = \begin{cases} z_i + 1, & \text{with probability } p, \\ z_i - 1, & \text{with probability } q. \end{cases} \tag{5.22}$$

Now, using the negative binomial distribution, we can model the probability of success of the adversary. First, as defined in [93], the probability of the attacker to surpass the blocks generated by the honest miners as:

$$Q_z = \left(\frac{q}{p}\right)^{z+1}, \tag{5.23}$$

where $z \geq 0$. We then assume a voter waits for n new blocks to be generated by the honest miners to be appended to the blockchain beyond the block containing their vote. We also assume that, at that time, the adversary is able to secretly generate $m = n + 1$ blocks. This can be modeled as m blocks that the adversary can generate before n blocks are generated by the honest miners. Therefore, the probability of reversing a transaction for a given value m is:

$$\mathsf{Pr}(m) = \binom{m+n-1}{m} \times p^n q^m. \tag{5.24}$$

The probability for the adversary to surpass successfully the number of blocks generated by the honest miners can be computed as:

$$\begin{aligned}\mathsf{Pr}_s &= \sum_{m=0}^{n-1} \mathsf{Pr}(m) \times Q_{n-m-1} \\ &= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{m} \times p^n q^m \times \left(1 - \left(\frac{q}{p}\right)^{n-m}\right) \\ &= 1 - \sum_{m=0}^{n-1} \binom{m+n-1}{m} \times (p^n q^m - p^m q^n).\end{aligned} \tag{5.25}$$

For attackers to succeed in performing the attack, they must be able to generate more blocks, i.e. $m > n$. The likelihood decreases as n or p increase since $q < p$ and the attackers are limited to the election time frame. Therefore, as more blocks are appended to the blockchain, manipulation of the election result becomes less likely or even infeasible. $\square$

5.5 Performance Evaluation

In this section, we present a performance evaluation for d-CRAME and compare it with two blockchain-based schemes presented in [36] and [40]. We specifically select these schemes to analyze the difference in computational complexity when applying different cryptographic primitives. We formulate the computational costs of each voting stage for these three schemes based on the number of multiplication (M) and exponentiation (E) operations. Table 5.1 summarizes the number of these two operations for each scheme.

Table 5.1: d-CRAME performance comparison.

	Setup						**Voter Reg.**						**Acquiring Ballots**						**Casting Votes**						**Tabulation**					
Scheme	[36]		[40]		d-CRAME		[36]		[40]		d-CRAME		[36]		[40]		d-CRAME		[36]		[40]		d-CRAME		[36]		[40]		d-CRAME	
Operation	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E	M	E
Voter	0	0	0	0	0	0	1	2	1	1	0	1	0	0	n/a		2	2	2	3	$2(m+b+1)$	$3m+b+9$	2	2	0	0	0	0	0	0
Moderator	n/a		2	0	0	1	n/a		0	0	0	0	n/a		n/a		$2n$	$6n$	n/a		0	0	0	0	n/a		10	5	0	0
Registrar	n/a		$5+\kappa$	$1+2\kappa$	0	1	n/a		0	0	$2n$	n	n/a		n/a		0	2n	n/a		0	0	0	0	n/a		n/a		0	0
Tallier	n/a		n/a		0	0	n/a		n/a		0	0	n/a		n/a		0	0	n/a		n/a		0	0	n/a		n/a		$2n$	$2n$
Smart Contract	0	0	0	0	0	0	0	0	0	n	0	0	$2n$	$2n$	n/a		0	0	0	0	$n(1+b)$	$5n$	0	0	n	0	1	$3+m$	0	0

5.5.1 Setup

In [36], the setup consists of an admin that authenticates voters with their Ethereum user-controlled accounts then updates a whitelist of voters to include all eligible voters. However, the paper did not discuss any cryptographic operations in the authentication process. We assume that the setup for this scheme does not require any M or E operations.

On the contrary, [40] requires a smart contract admin to generate an RSA key pair (sk_S, pk_S), imposing 2M operations. This key pair is used to sign/verify transactions between voters and the smart contracts. The scheme also requires a voting admin to generate the homomorphic Paillier key pair $(sk_{\mathsf{Paillier}}, pk_{\mathsf{Paillier}})$ at a total computational cost of 4M and a single E operation. Using the generated key, the voting admin next encrypts the value zero κ times and stores all encryptions as set T. The set T is used to enforce randomness as each voter is randomly assigned a value from the set during vote casting that is used to prevent coercion. Each encryption in set T imposes a single M and 2E operations, resulting in a total of $(\mathsf{M} + 2\mathsf{E})\kappa$ operations during encryption. Finally, the voting admin generates a Short Linkable Ring Signatures (SLRS) [94] key pair which requires one M operation. This key pair is used during the election to prevent linking voters to their ballots. For explanation purposes, we refer to the smart contract admin as the moderator and the voting admin as the registrar. In comparison to both schemes, d-CRAME requires the moderator and registrar to generate their public key pairs, y_m and y_r, that are used during the entire election process, imposing a single E operation for each.

5.5.2 Voter Registration

At the voter registration stage, [36] requires each voter to generate a key pair (sk_i, pk_i) at a cost of one E operation. Voters must also generate a non-interactive Zero-Knowledge Proof $\mathsf{ZKP}(x_i)$ to prove knowledge of their secret key sk_i. Specifically, it uses a Schnorr proof [95] made non-interactive using the Fiat-Shamir heuristic [96], resulting in an additional M and E operation. Once derived, voters broadcast pk_i and $\mathsf{ZKP}(x_i)$ through the specified election smart contract.

In [40], voter registration requires interaction between the voters and an election smart contract. Initially, eligible voters obtain the SLRS parameters generated during the setup phase. Using these parameters, voters then generate their SLRS key pairs (sk_i, pk_i), imposing a single M operation each. Voters then send their public keys pk_i to the blockchain through the smart contract. In order for the smart contract to accept and register their public keys, voters must sign the transaction with the RSA key generated during the setup phase, imposing a single E operation. The smart contract then verifies the signature and adds the public keys to the blockchain if valid. Each verification requires a single E operation. As mentioned in [40], this transaction verification process is performed with every interaction between a voter and a smart contract throughout the entire election.

In comparison, d-CREAM relies on a registrar to facilitate voter registration. Voters begin by generating their key pairs (sk_i, pk_i). After verifying the eligibility of voters, the registrar signs their public keys and adds them to the electoral roll. The total signatures for registering all voters are $2n\mathsf{M}$ and $n\mathsf{E}$ operation.

5.5.3 Acquiring Ballots

To acquire a ballot, [36] initially requires the election smart contract to verify all n received $\mathsf{ZKP}(x_i)$. A single verification costs one M and $2\mathsf{E}$ operations. Next, the smart contract generates all n digital ballots, where a single ballot generation requires a total of $n\mathsf{M}$ operations. Voters acquire their corresponding generated ballots and use them in the next stage during vote casting.

In [40], voters are not required to perform any computations to acquire a digital ballot. Instead, they can immediately cast their votes once registration is complete.

In comparison to both schemes, d-CRAME involves voter engagement with the moderator and registrar. Initially, the registrar generates the set of random and digitally signed ballots. Next, the voters interact with the moderator, who obscures their identities from the registrar while facilitating ballot distribution. The total operations performed by a voter, the moderator, and the registrar are $2\mathsf{M} + 2\mathsf{E}$, $2n\mathsf{M} + 6n\mathsf{E}$, and $2n\mathsf{E}$ operations, respectively.

5.5.4 Casting Vote

To cast a vote, [36] requires each voter to perform one M and $2\mathsf{E}$ operations. Each voter must also generate another $\mathsf{ZKP}(x_i)$ to prove that they have voted either yes or no. The generated $\mathsf{ZKP}(x_i)$ requires an additional M and E operation.

In [40], voters can vote for multiple candidates, however, casting votes is a more complex process. Initially, voters select their desired candidates within an eligible pool of candidates and encode it. Next, they homomorphically encrypt the result using their previously generated Paillier secret key. These two functions together impose $3\mathsf{M}$ and $2\mathsf{E}$ operations. Following that, the voters compute a Proof of Knowledge (PoK) to prove that they correctly encrypt a candidate within the eligible pool. This proof requires $2m\mathsf{M}$ and $(3m+1)\mathsf{E}$ operations, where m is the total number of running candidates, which could be a big number for a large scale election. Once computed, the voters send their encryption and derived PoKs to the smart contract for verification. The computational cost of a single verification imposes $2\mathsf{M}$ and $3\mathsf{E}$ operations. If valid, the smart contract signs the result and returns it to the voter. As previously discussed, a single signature imposes a single E operation for each voter. Upon receiving it, voters validate the signature at a computation cost of one E operation. If valid, the voters generate SLRS over the result to finalize their votes and cast it to the blockchain. A single SLRS requires voters to initially generate a witness value, imposing $(2b-1)\mathsf{M}$ and $b\mathsf{E}$ operations, where b represents the number of ring members, which has to be a reasonably big number. Next, they generate a proof of knowledge for their signatures, imposing $5\mathsf{E}$ operations. Once the SLRS is generated, the voters send it to the smart contract for verification. In order for the smart contract to verify a single SLRS, it imposes $(b-1)\mathsf{M}$ and one E operation. Finally, if valid, a vote is permanently stored on the blockchain.

In comparison, d-CRAME requires a voter to double encrypt bal_i and attach it to the Vote_i where, Vote_i is a sequence of bits representing each candidate, 1 if voting for cand_k and 0 if not voting for cand_k. The total operations performed by a voter include $2\mathsf{M}$ and $2\mathsf{E}$ operations.

5.5.5 Tabulation

In [36], tabulation requires verifying each $\mathsf{ZKP}(x_i)$ submitted the voters, imposing one M and $2\mathsf{E}$ operations per verification. Next, the tally is computed requiring $n\mathsf{M}$ operations.

In [40], tabulation is performed over multiple steps. The election smart contract first adds all published and encrypted votes then signs the result and sends it to the admin, imposing one E operation. The admin next verifies the signature at a cost of one E operation and begins to homomorphically decrypt the sum. Homomorphic decryption enforces a computational cost of 3M and one E operations. The admin must also perform a decryption and verify the correctness of the PoK and sends the results back to the smart contract, resulting in 7M and 3E operations. The smart contract verifies the correctness of the information sent imposing one M and 2E operations. If valid, the smart contract computes the ballots for m candidates, where each candidate requires one E operation.

In comparison to both schemes, for d-CRAME, the tallier is required to double decrypt each existing vote using the revealed moderator and registrar private keys, x_m and x_r. This imposes a computational cost of $2n\mathsf{M}$ and $2n\mathsf{E}$ operations.

5.6 Empirical Results

In this section, we present our empirical results of various simulations of d-CRAME. We implement two versions of d-CRAME, a desktop application, and a smartphone application. The desktop application is implemented using Maple v16 and we simulate it on a MacBook Air running OS X 10.13.6 equipped with 2 cores, 1.8 GHz Intel Core i5, and 8 GB 1600 MHz DDR3. The smartphone application is developed over Xcode version 11.2.1 and is written in Swift 5. We use the BigInt library [97] to perform large number cryptographic computations. Our smartphone simulations are performed over an iPhone XR running iOS 13.2.2 equipped with the Apple A12 Bionic, a 64-bit ARMv8.3-A six-core CPU, with two cores running at 2.49 GHz.

Both applications interact with our Solidity-based smart contract that we deploy over the Ethereum Ropsten testnet blockchain to cast the encrypted votes at the end of the voting stage. The main inputs to the smart contract are the two encrypted vote components, $B_i = (c_3, c_4)$, as depicted in equation (5.13). For our desktop application, we utilize MetaMask [98], a browser plug-in that allows voters to manage their Ethereum wallets and call our deployed smart contract to cast their votes. For our smartphone application, we incorporate the web3swift library [99] into our code to provide the voters with the same functionality to cast their votes using their mobile devices. We note that the performance relies on the selection of software packages. Our selection does not guarantee the best performance. Instead, it shows that *d*-CRAME is suitable for large-scale elections.

For an election, *d*-CRAME is run by each voter on either a desktop or mobile device. After vote casting is complete, tabulating the votes is a job performed by the tallier, which is generally performed offline using powerful computers. We, therefore, assume that all stages are performed over a mobile device while tabulation is performed by anyone that has access to a desktop machine. In comparison to other schemes, *d*-CRAME is the only scheme that allows voters to generate the election results themselves if they wish and can afford the required computational requirements.

Therefore, we focus our analysis on investigating the different time costs for casting a vote for both a computer and a smartphone. Specifically, we measure the time costs to perform the double encryption computation presented in equation (5.13). Fig. 5.3 shows the time costs we obtain under eight different encryption key sizes. The presented time costs are the average time costs of ten different trials for each key size.

Based on our results, we come to various conclusions. Fig. 5.3(a) shows that with maximized security at a key size of 4096 bits, voters can cast their votes in approximately 0.11 seconds using a desktop. Correspondingly, Fig. 5.3(b) shows that to maintain the same level of security while running *d*-CRAME over a smartphone, it would take less than a minute to

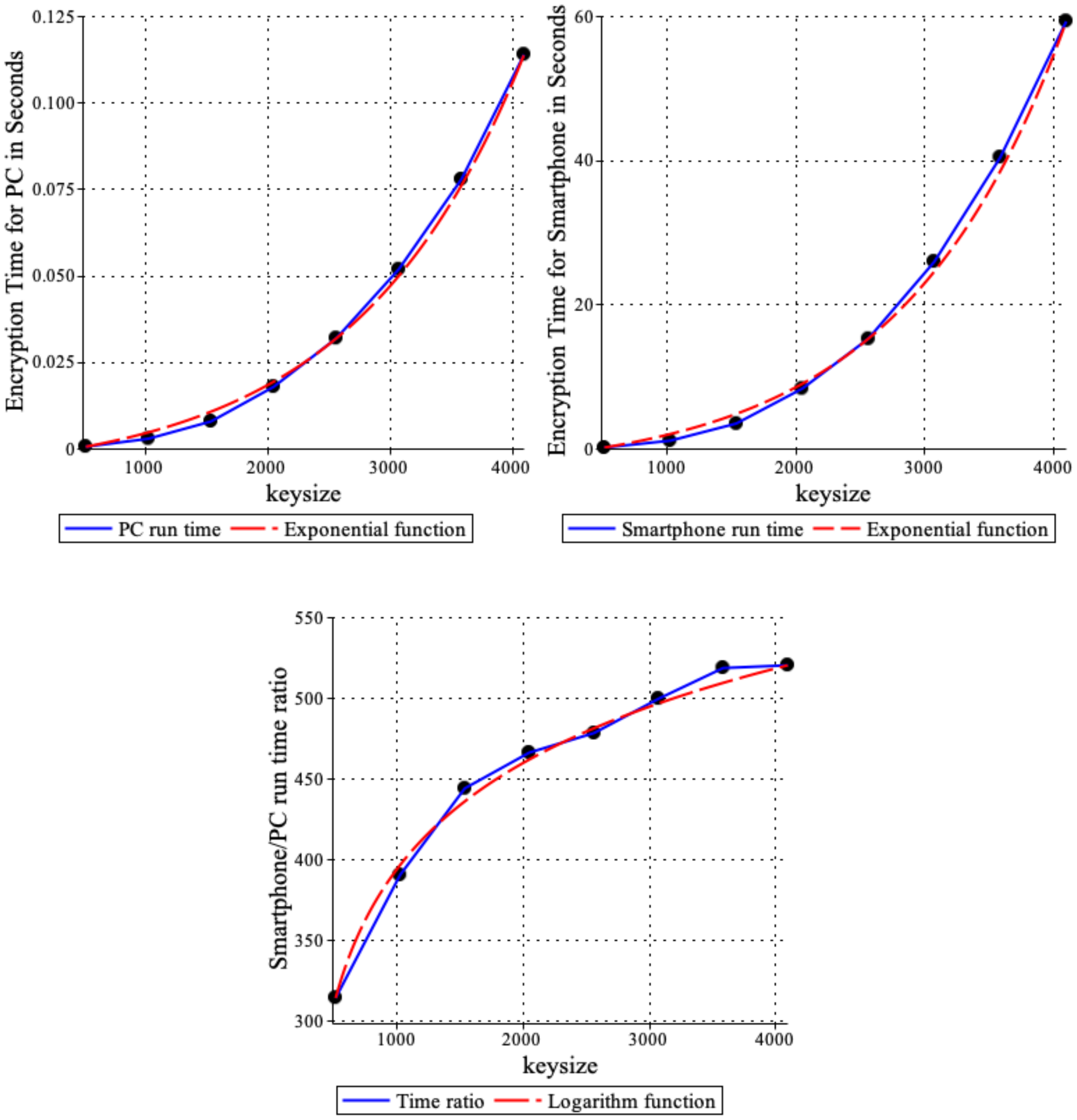


Figure 5.3: *d*-CRAME time comparison for various key sizes: (a) in computer, (b) in smartphone, (c) smartphone/computer.

cast a vote. In both cases, the time increases exponentially with increasing of key sizes. Today, in practice, key sizes of 2048 bits are generally considered secure, hence, casting a vote through a mobile device may even be reduced to approximately 8.36 seconds. The difference between the time costs and of running *d*-CRAME over a desktop and a smartphone is evident because of the processor capabilities we describe above. Desktop machines are generally built with more powerful processors giving them a conspicuous advantage over smartphones. However, the purpose of this analysis is to prove that even with this advantage, it is still feasible to run *d*-CRAME over a smartphone and achieve acceptable results.

To further analyze our findings, we also measure the smartphone to desktop time cost ratio and observe its change as we increase the key size. Fig. 5.3(c) presents these results showing that as the key size increases, the ratio increases logarithmically. This suggests that, beyond a certain key size, the advantage of running *d*-CRAME over a desktop versus running it over a smartphone decays as the keys grow in size. For example, as shown in Fig. 5.3(c), the smartphone/desktop ratio at a key size of 2048 bits is 466 and increases to 478 with key size of 2560 bits, showing a 2.5% increase. This increase is smaller when compared to the ratio increase between key size of 1536 bits, where the ratio is 444, and that at key size of 2048 bits, showing an increase of 5%. This pattern can be observed between all key sizes shown in the figure.

5.7 Summary

In this chapter, we proposed *d*-CRAME, a novel blockchain-based and remote electronic voting scheme. The proposed scheme is designed to run large-scale elections and aims at improving voter turnout. Our security and privacy analyses show that *d*-CRAME is secure, preserves voter privacy, protects voters against coercers, and maintains the integrity of election results. In our performance analysis, we compare the computational complexity of d-CRAME to two schemes and show that d-CRAME is more appropriate for large-scale elections. Finally, in our empirical results, we present the results of running various simulations for d-CRAME over both a desktop machine and a smartphone. Our results show that it is feasible to run d-CRAME over a mobile device, hence improving the accessibility of elections.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, we presented sensitive data sharing schemes that address data sharing in both centralized and distributed systems. In the first scheme discussed, P-MOD, the data owner partitions a data file into multiple segments and shares the independently encrypted segments in a privilege-based access structure. We formally proved the security of this scheme against an adaptively chosen-plaintext attack assuming the DBDH assumption holds. We then ran simulations of P-MOD to compare its performance to CP-ABE and FH-CP-ABE. P-MOD demonstrated an 88.3% improvement in key generation time-cost compared to these schemes. Similarly, P-MOD outperformed these schemes in encryption time-cost by approximately 78.4% and 18%, respectively, and in decryption time-cost by approximately 85.4% for both schemes.

Following P-MOD, we explored blockchain and distributed systems to evaluate the benefits of blockchain and the security pitfalls. We delved into the underlying structure and security of the first system to adopt blockchain, Bitcoin. To explore its security, we analyzed double-spending attacks and evaluated the probability of success of such attacks, given the computing power of the attackers. Our probability analysis concluded that there is a trade-off between the time waited before accepting a transaction and the profit attackers could gain. Through understanding the security limitations of Bitcoin, we gained an understanding of how to enhance the security of a distributed system.

In the second scheme presented, *d*-MABE, we applied the concepts of privilege-based multilevel data sharing to a distributed system by using blockchain and smart contracts. We used electronic medical records (EMRs) to demonstrate the benefits of such a system. We implemented smart contracts to facilitate access verification and key granting. In the security and performance analysis sections, we showed that d-MABE is both secure and practical.

In the third scheme, *d*-CRAME, we proposed a novel remote electronic voting scheme. The proposed scheme is designed to run large-scale elections and aims at improving voter turnout. Our security and privacy analyses showed that *d*-CRAME is secure, preserves voter privacy, protects voters against coercers, and maintains the integrity of election results. In our performance analysis, we compared the computational complexity of *d*-CRAME to two schemes and show that *d*-CRAME is more appropriate for large-scale elections. Finally, in our empirical results, we implemented *d*-CRAME and presented the results of running various simulations for over both a desktop machine and a smartphone. Our results show that it is feasible to run *d*-CRAME over a mobile device, hence improving the accessibility of elections and overall voter turnout.

The schemes developed and presented in this dissertation aim to provide efficient, secure, and privacy-preserving methods to share sensitive data. They share a centralized goal, empowering the data owners to define how their sensitive data will be shared with data users. The benefits of such systems can be seen in many applications. Electronic medical records were initially chosen as a common theme in this dissertation since electronic health data often contain the most sensitive information of a person. The health sector is also one of the most targeted in terms of malicious attacks. The schemes presented address the security concerns seen today and provide an alternative method that can provide further advantages. We then followed with an electronic voting scheme to show that data censorship is also an imperative factor that must be considered and is applicable to other domains.

6.2 Future Work

There is continued interest in building data sharing and censorship schemes that mitigate the security and privacy risks of trusted third parties. These security risks include preventing data loss and leakage of sensitive data, protecting data integrity, and ensuring its availability and verifiability whenever requested. In the following, we highlight some promising research directions that incorporate the security and privacy advantages of distributed networks and build on the work presented in this dissertation.

- *Improving the privacy of distributed storage.* Distributed systems such as IPFS can provide efficient methods for data storage to be accessed by data users. In such systems, data is stored across a P2P network, hence, eliminating single points of failure. In addition, distributed storage systems can help improve the efficiency of retrieving data when requested. However, these systems lack sufficient privacy countermeasures. Users may protect the privacy of their data by encrypting it before uploading it to the distributed storage, however, this results in additional computational and monetary costs. We plan on performing more research in this area to improve the privacy countermeasures and reducing the overall costs of using distributed storage. We plan to investigate the feasibility of a distributed storage system as an alternative to current centralized systems with similar or superior security and privacy countermeasures.

- *Improving linkability/traceability of published data.* We have shown that distributed platforms may significantly help improve security and single points of failure issues when compared to reliance on a trusted third party. However, distributed platforms also introduce new privacy challenges that must be addressed in order for these systems to be accepted. Sharing data in a distributed platform exposes data owners to loss of their anonymity. Research in mixed networks or use of expensive cryptographic primitives such as ring signatures [100] and zero-knowledge proofs [101] have been introduced to overcome this issue. However, it has been proven that mixing and the current cryptographic primitives solutions cannot provide perfect secrecy [102] and are expensive, making the proposed solutions infeasible to be adopted in real-life. This becomes a bigger issue if an application requires user or universal verifiability where individuals are allowed to verify that their data is stored correctly at all times. We wish to explore potential solutions to this privacy issue that will allow users to publicly share their data in a distributed manner while continuously maintaining their privacy. This would be beneficial to systems such as electronic voting schemes where voters could continue to keep their votes secret even after the election results have been announced.

- *Designing data sharing and censorship schemes for expanded application domains.* Since the outset of distributed platforms such as blockchain, it has become an interesting research area to design distributed data sharing and censorship schemes. While blockchain was originally invented to enable the decentralized digital currency Bitcoin, in this dissertation we utilized this technology to design schemes for secure data sharing of electronic medical records and censorship of votes in electronically held elections. Similarly, several other domains may make use of the permanent, transparent, and irreversible features of blockchain. One particular area of interest is the application of blockchain in commercial operations and supply chain of manufacturing. In today's world, materials can be acquired from multiple vendors spanning across continents. Full immutable records on the history of all materials into a finished good can allow deeper data analysis and trending capabilities. Here we discuss two industries, the automobile and pharmaceutical industry, in which blockchain could be leveraged to benefit consumers and companies.

In the automobile industry, recalls can cause consumer distrust. In extreme cases, faulty parts may be dangerous or lead to fatal outcomes. By utilizing blockchain to create a record on the parts, manufacturing history and ownership history of a vehicle, recall alerts can be expedited to consumers. In addition, vehicle manufacturers will have access to data on parts and manufacturing history that they can leverage to find the root cause of issues. This data could even lead to the prevention of a recall by tracing a faulty part to vehicles still in manufacture, or identification of a certain supplier providing a high number of faulty parts. In another application in the automobile industry, a public, immutable ledger may give rise to shared public ownership of vehicles. Blockchain can be utilized to make small payments on vehicle use as needed. In addition, a fully traceable history of the vehicle can allow preventive maintenance to be maintained without a sole owner.

Similar to the automobile industry, medicine recalls in the pharmaceutical industry can also cause consumer distrust and lead to dangerous outcomes. However, this industry presents an even greater challenge for traceability because a consumer may receive the finished product in the form of tablets dispensed by a pharmacist, with no indication to the consumer if their product is affected by a recall after. By leveraging blockchain, the pharmaceutical industry can work with pharmacists to create an end-to-end history of the materials and manufacturing of a product that can be used to inform patients sooner of product recalls.

In the future, we will explore the applicability and feasibility of distributed platforms such as blockchain to other domains.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Healey, "Umass memorial health care entities to pay $230,000 to resolve ag's lawsuit over data breaches," https://www.mass.gov/news/umass-memorial-health-care-entities-to-pay-230000-to-resolve-ags-lawsuit-over-data-breaches, 2018, [Online; accessed 10-December-2018].

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2005, pp. 457–473.

[3] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 195–203.

[4] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security.* Acm, 2006, pp. 89–98.

[5] N. Attrapadung, B. Libert, and E. De Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *International Workshop on Public Key Cryptography.* Springer, 2011, pp. 90–108.

[6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07).* IEEE, 2007, pp. 321–334.

[7] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," *arXiv preprint arXiv:0903.2171*, 2009.

[8] L. Cheung and C. Newport, "Provably secure ciphertext policy abe," in *Proceedings of the 14th ACM conference on Computer and communications security.* ACM, 2007, pp. 456–465.

[9] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *2010 Int. Conf. on the Theory and Applications of Cryptographic Techniques.* Springer, 2010, pp. 62–91.

[10] L. Ibraimi, M. Petkovic, S. Nikova, P. Hartel, and W. Jonker, "Mediated ciphertext-policy attribute-based encryption and its application," in *International Workshop on Information Security Applications.* Springer, 2009, pp. 309–323.

[11] S. Roy and M. Chuah, "Secure data retrieval based on ciphertext policy attribute-based encryption (cp-abe) system for the dtns," Citeseer, Tech. Rep., 2009.

[12] J. Lai, R. H. Deng, and Y. Li, "Fully secure cipertext-policy hiding cp-abe," in *International Conference on Information Security Practice and Experience.* Springer, 2011, pp. 24–39.

[13] J. Lai, R. Deng, and Y. Li, "Expressive cp-abe with partially hidden access structures," in *Proceedings of the 7th ACM symposium on information, computer and communications security*, 2012, pp. 18–19.

[14] F. Guo, Y. Mu, W. Susilo, D. S. Wong, and V. Varadharajan, "Cp-abe with constant-size keys for lightweight devices," *IEEE transactions on information forensics and security*, vol. 9, no. 5, pp. 763–771, 2014.

[15] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1214–1221, 2010.

[16] K. Emura, A. Miyaji, A. Nomura, K. Omote, and M. Soshi, "A ciphertext-policy attribute-based encryption scheme with constant ciphertext length," in *International Conference on Information Security Practice and Experience.* Springer, 2009, pp. 13–23.

[17] Z. Zhou and D. Huang, "On efficient ciphertext-policy attribute based encryption and broadcast encryption," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 753–755.

[18] J. Herranz, F. Laguillaumie, and C. Ràfols, "Constant size ciphertexts in threshold attribute-based encryption," in *International Workshop on Public Key Cryptography.* Springer, 2010, pp. 19–34.

[19] A. Lewko and B. Waters, "New proof methods for attribute-based encryption: Achieving full security through selective techniques," in *Annual Cryptology Conference.* Springer, 2012, pp. 180–198.

[20] C. Chen, Z. Zhang, and D. Feng, "Efficient ciphertext policy attribute-based encryption with constant-size ciphertext and constant computation-cost," in *International Conference on Provable Security.* Springer, 2011, pp. 84–101.

[21] Z. Liu, Z. Cao, and D. S. Wong, "White-box traceable ciphertext-policy attribute-based encryption supporting any monotone access structures," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 76–88, 2012.

[22] V. Goyal, A. Jain, O. Pandey, and A. Sahai, "Bounded ciphertext policy attribute based encryption," in *International Colloquium on Automata, Languages, and Programming.* Springer, 2008, pp. 579–591.

[23] A. Ge, R. Zhang, C. Chen, C. Ma, and Z. Zhang, "Threshold ciphertext policy attribute-based encryption with constant size ciphertexts," in *Australasian Conference on Information Security and Privacy.* Springer, 2012, pp. 336–349.

[24] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of the 17th ACM conference on Computer and communications security.* ACM, 2010, pp. 735–737.

[25] G. Wang, Q. Liu, J. Wu, and M. Guo, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers," *computers & security*, vol. 30, no. 5, pp. 320–331, 2011.

[26] C. Gentry and A. Silverberg, "Hierarchical id-based cryptography," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2002, pp. 548–566.

[27] R. Bobba, H. Khurana, and M. Prabhakaran, "Attribute-sets: A practically motivated enhancement to attribute-based encryption," in *European Symposium on Research in Computer Security.* Springer, 2009, pp. 587–604.

[28] Z. Wan, J. Liu, and R. H. Deng, "Hasbe: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE transactions on information forensics and security*, vol. 7, no. 2, pp. 743–754, 2012.

[29] H. Deng, Q. Wu, B. Qin, J. Domingo-Ferrer, L. Zhang, J. Liu, and W. Shi, "Ciphertext-policy hierarchical attribute-based encryption with short ciphertexts," *Information Sciences*, vol. 275, pp. 370–384, 2014.

[30] J. Li, Q. Wang, C. Wang, and K. Ren, "Enhancing attribute-based encryption with attribute hierarchy," *Mobile networks and applications*, vol. 16, no. 5, pp. 553–561, 2011.

[31] S. Wang, J. Zhou, J. K. Liu, J. Yu, J. Chen, and W. Xie, "An efficient file hierarchy attribute-based encryption scheme in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1265–1277, 2016.

[32] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE.* IEEE, 2015, pp. 180–184.

[33] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on.* IEEE, 2016, pp. 25–30.

[34] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[35] A. Dubovitskaya, Z. Xu, S. Ryu, M. Schumacher, and F. Wang, "Secure and trustable electronic medical records sharing using blockchain," in *AMIA Annual Symposium Proceedings*, vol. 2017. American Medical Informatics Association, 2017, p. 650.

[36] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *International Conference on Financial Cryptography and Data Security.* Springer, 2017, pp. 357–375.

[37] F. Hao, P. Y. Ryan, and P. Zieliński, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, no. 2, pp. 62–67, 2010.

[38] G. G. Dagher, P. B. Marella, M. Milojkovic, and J. Mohler, "Broncovote: Secure voting system using ethereum's blockchain," 2018.

[39] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-voting with blockchain: an e-voting protocol with decentralisation and voter privacy," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* IEEE, 2018, pp. 1561–1567.

[40] B. Yu, J. K. Liu, A. Sakzad, S. Nepal, R. Steinfeld, P. Rimba, and M. H. Au, "Platform-independent secure blockchain-based voting system," in *International Conference on Information Security.* Springer, 2018, pp. 369–386.

[41] B. Wang, J. Sun, Y. He, D. Pang, and N. Lu, "Large-scale election based on blockchain," *Procedia Computer Science*, vol. 129, pp. 234–237, 2018.

[42] M. Lichman, "Uci machine learning repository," = http://archive.ics.uci.edu/ml, 2013, [Online; accessed 9-September-2016].

[43] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *International Workshop on Public Key Cryptography.* Springer, 2011, pp. 53–70.

[44] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard.* Springer Science & Business Media, 2013.

[45] J. Katz and Y. Lindell, *Introduction to modern cryptography.* Chapman and Hall/CRC, 2014.

[46] J. Wang, "Ciphertext-policy attribute based encryption java toolkit," https://github.com/junwei-wang/cpabe, 2015, [Online; accessed 9-September-2016].

[47] A. De Caro and V. Iovino, "Jpbc: Java pairing based cryptography," in *Computers and communications (ISCC), 2011 IEEE Symposium on.* IEEE, 2011, pp. 850–855.

[48] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography.* Springer, 1990, pp. 437–455.

[49] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the Theory and Application of Cryptographic Techniques.* Springer, 1987, pp. 369–378.

[50] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," in *Sequences II.* Springer, 1993, pp. 329–334.

[51] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008, [Online; accessed 12-December-2017].

[52] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.", 2014.

[53] "Satoshi client node discovery," https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery, 2017, [Online; accessed 15-December-2017].

[54] D. R. L. Brown, "SEC 2: Recommended elliptic curve domain parameters," Certicom Research, Tech. Rep., 2010.

[55] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems.* Springer, 2002, pp. 251–260.

[56] M. Rosenfeld, "Analysis of bitcoin pooled mining reward systems," *arXiv preprint arXiv:1112.4980*, 2011.

[57] S. Pool, "Reward system," https://slushpool.com/help/manual/rewards, 2017, [Online; accessed 12-December-2017].

[58] T. B. C. developers, "bitcoin/bitcoin," https://github.com/bitcoin/bitcoin, 2017, [Online; accessed 26-March-2017].

[59] A. Loibl, "Namecoin," *Network Architectures and Services*, vol. 107, 2014.

[60] "Bitcoin Cash," https://www.bitcoincash.org/, 2017, [Online; accessed 10-October-2017].

[61] J. Hilliard, "Reduced threshold segwit masf," https://github.com/bitcoin/bips/wiki/Comments:BIP-0091, 2017, [Online; accessed 13-November-2017].

[62] P. Wuille, "Segregated witness and its impact on scalability," in *SF Bitcoin Devs Seminar*, 2015.

[63] H. Finney, "Best practice for fast transaction acceptance - how high is the risk?'," https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384, Feb. 2011, [Online; accessed 13-November-2017].

[64] K. Sigman, "Gambler's Ruin Problem," http://www.columbia.edu/~ks20/FE-Notes/4700-07-Notes-GR.pdf.

[65] A. P. Ozisik and B. N. Levine, "An explanation of nakamoto's analysis of double-spend attacks," *arXiv preprint arXiv:1701.03977*, 2017.

[66] M. Rosenfeld, "Analysis of hashrate-based double spending," *arXiv preprint arXiv:1402.2009*, 2014.

[67] "Mining hardware comparison," https://en.bitcoin.it/wiki/Mining_hardware_comparison, 2017, [Online; accessed 10-October-2017].

[68] “Non-specialized hardware comparison,” https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison, 2017, [Online; accessed 10-October-2017].

[69] Bitmain, “BIP process, revised,” https://www.antpool.com/, 2017, [Online; accessed 9-October-2017].

[70] B. China, “Bttc,” https://www.btcc.com/, 2011-2017, [Online; accessed 9-October-2017].

[71] L. HK Bixin Network Technology Co., “Bixin,” https://bixin.com/, [Online; accessed 9-October-2017].

[72] Bitmain, “Btc.com,” https://btc.com/, 2017, [Online; accessed 9-October-2017].

[73] Btctop, “btc.top,” http://btc.top/, 2017, [Online; accessed 9-October-2017].

[74] “U.S. energy information administration,” https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a, [Online; accessed 9-October-2017].

[75] E. Zaghloul, K. Zhou, and J. Ren, “P-mod: Secure privilege-based multilevel organizational data-sharing in cloud computing,” *arXiv preprint arXiv:1801.02685*, 2018.

[76] W. Stallings, *Cryptography and network security: principles and practice.* Pearson Upper Saddle River, NJ, 2017.

[77] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, “Secure attribute-based systems,” *J. of Comp. Security*, vol. 18, no. 5, pp. 799–837, 2010.

[78] A. Singh, T. wan Ngan, P. Druschel, and D. S. Wallach, “Eclipse attacks on overlay networks: Threats and defenses,” in *In IEEE INFOCOM.* Citeseer, 2006.

[79] “Solidity,” http://solidity.readthedocs.io/en/v0.4.24/, 2018.

[80] “Ropsten (revival) testnet,” https://ropsten.etherscan.io, 2018.

[81] California Legislative Information, “Chapter 1. patient access to health records [123100 - 123149.5],” http://leginfo.legislature.ca.gov/faces/codes_displaySection.xhtml?lawCode=HSC&sectionNum=123110, 2017.

[82] “Maximum fees allowed for providing health care information,” https://www.utmb.edu/him/pdf/fee_schedule.pdf, 2017.

[83] "395.3025-patient and personnel records; copies; examination," http://www.flsenate.gov/laws/statutes/2011/395.3025, 2018.

[84] "Statute sections 17 and 18 of public health law (phl)," https://www.health.ny.gov, 2018.

[85] "Copying fees adjustments," https://illinoiscomptroller.gov/agencies/resource-library/statutorily-required/copying-fees-adjustments/, 2018.

[86] "Json rpc," https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_estimategas, 2018.

[87] "Ethereum average gasprice chart," https://etherscan.io/chart/gasprice, 2018.

[88] S. E. White, "A review of big data in health care: challenges and opportunities," *Open Access Bioinformatics*, vol. 6, pp. 13–18, 2014.

[89] National Conference of State Legislatures, "Double voting," http://www.ncsl.org/research/elections-and-campaigns/double-voting.aspx, 2018.

[90] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Proceedings of the 2005 ACM workshop on Privacy in the electronic society.* ACM, 2005, pp. 61–70.

[91] K. Sako and J. Kilian, "Receipt-free mix-type voting scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 1995, pp. 393–403.

[92] D. W. Jones, "Some problems with end-to-end voting," in *End-to-End Voting Systems Workshop, Washington DC: National Institute for Standards and Technology (NIST), at www. divms. uiowa. edu/˜ jones/voting/E2E2009. pdf [last accessed December 23, 2011]*, 2009.

[93] E. Zaghloul, T. Li, M. Mutka, and J. Ren, "Bitcoin and blockchain: Security and privacy," *arXiv preprint arXiv:1904.11435*, 2019.

[94] M. H. Au, S. S. Chow, W. Susilo, and P. P. Tsang, "Short linkable ring signatures revisited," in *European Public Key Infrastructure Workshop.* Springer, 2006, pp. 101–115.

[95] C.-P. Schnorr, “Efficient signature generation by smart cards,” *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[96] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Conference on the Theory and Application of Cryptographic Techniques.* Springer, 1986, pp. 186–194.

[97] “Arbitrary-precision arithmetic in pure swift,” https://github.com/attaswift/BigInt, 2019, [Online; accessed 15-May-2019].

[98] “Metamask browser extension,” https://github.com/MetaMask/metamask-extension, 2019, [Online; accessed 15-May-2019].

[99] “Elegant web3js functionality in swift. native abi parsing and smart contract interactions.” https://github.com/matter-labs/web3swift, 2019, [Online; accessed 15-May-2019].

[100] T. Ruffing, P. Moreno-Sanchez, and A. Kate, “Coinshuffle: Practical decentralized coin mixing for bitcoin,” in *European Symposium on Research in Computer Security.* Springer, 2014, pp. 345–364.

[101] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized anonymous payments from bitcoin,” in *Security and Privacy (SP), 2014 IEEE Symposium on.* IEEE, 2014, pp. 459–474.

[102] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, “An empirical analysis of traceability in the monero blockchain,” *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 143–163, 2018.