Securing Data with Provenance and Cryptography

アムリル, シャリム

https://hdl.handle.net/2324/1959201

出版情報：Kyushu University, 2018, 博士（情報科学）, 論文博士
バージョン：
権利関係：

Securing Data with Provenance and Cryptography

Amril Syalim

Department of Informatics
Kyushu University

Abstract

With the advances in the network technology, it is now possible to implement the databases and applications as services. The main advantage of this model is the users can use the services at a fraction of the cost to maintain their own servers. The users can also use a powerful distributed computational resource that is provided as a service for their computation heavy tasks. However, this model has a fundamental problem, because the data are stored in the servers owned by the entities that are not controlled by the users, the users need to concern about the confidentiality and integrity of their sensitive data.

In a distributed system, because the tasks can be executed by many computers, some auditors may need to verify the integrity of data produced by the system. Many researchers suggested the idea to implement the provenance concept in the distributed systems. In the context of the computer systems, the provenance of data is recorded as a collection of assertions created by the process executors that describe the origins and the processes to produce the data. The provenance is stored in a special database, we call the Provenance Store, that should be accessible to the auditors who need to verify the data integrity.

The Provenance Store should be protected from malicious entities who try to update the provenance assertions. The update to the provenance causes the integrity problems, namely "inconsistent claims" and "inconsistent interpretations" problems. Storing the provenance in a trusted storage can prevent the attack. However, it is not practical to be implemented in many systems. We propose an integrity scheme that can be used to detect any changes to the provenance assertions by employing a signature chain and assigning a consecutive counter produced by a Trusted Counter Server (TCS) to each assertion.

The provenance should also be protected from unauthorized accesses. The existing methods for the access controls are designed for regular data that are not suitable for the provenance. A critical information in the provenance that needs to be protected by the access control is the causal relationships between the process and the data. We propose a method to implement access control system by defining the access right we call TRACE, that can be used to define access policy to a collection of assertions that have causal relationships to a specific assertion. We combine the TRACE right with a Multilabels method to support better granularity of the access restrictions.

In this thesis, we also discuss the method to protect the integrity of a sequence of documents by using digital signature. The signature is used to prove the authenticity of each document and the order of the documents in the sequence. The existing signature schemes have some disadvantages: either we need to include another information to prove the order of the sequence (i.e., trusted time-stamps/counters) or during the verification, we need to have access to all (or large numbers) of the signed documents. We propose a scheme that allows a party to sign a sequence of digital documents with the following characteristics: (1) the party can prove the order of the document in the sequence without having a trusted timestamps/-counters, (2) the party can verify the authenticity of the members of the sequence without having access to all other members in the sequence, and (3) the storage that is needed for the signature is smaller than signing each member of the sequence.

To protect confidentiality of the data in an untrusted server, the data owner can encrypt the data before storing the data in the server. The problem is whenever the data owner needs to update the encryption key, the data owner needs to re-encrypt the data by downloading the data from the server, decrypting the data, encrypting the data with the new key and uploading the new encrypted data to the server. It is desirable to have more efficient re-encryption method where the data owner can securely delegate the re-encryption process to a semi-trusted party (i.e., a proxy). Most symmetric ciphers do not support proxy encryption because malleability (the ability to meaningfully convert the ciphertext) is not a desired property in a secure encryption scheme. We propose a symmetric encryption scheme that supports proxy re-encryption by first transforming the plaintext into a random sequence of blocks using a variant of an All or Nothing Transform (AONT), and then transforming the random sequences by using some combinations of permutations.

Acknowledgements

I am indebted to my research advisors, Professor Kouichi Sakurai, Professor Yoshiaki Hori and Professor Takashi Nishide for their guidance, comments, supports and helps in everything related to my research work.

I am very grateful to my external advisors, Professor Toshihiro Yamauchi (Okayama University) and Dr. Naohiko Uramoto (IBM Tokyo Research Laboratory) for their advices during my study.

I am also very grateful to the Japanese Government (Monbukagakusho/MEXT) for the scholarship scheme and many supports provided by the Kyushu University during my study.

Many thanks to Professor Masafumi Yamashita and Professor Tsuyoshi Takagi for thoroughly checking my thesis and presentation that result in significant improvements to the thesis and presentation from their earlier drafts.

I would like to thank Professor Daisuke Ikeda and Professor Masaya Yasuda for their insightful and critical comments during the review of the thesis.

I would also like to express my gratitude to all members of Sakurai Lab, including all of the Japanese students, China Scholarship Council (CSC) students, Erwan Le Malécot, Chunhua Su, Prof. Kyung Hyune Rhee and all other members who have helped me with rich discussions during seminars or casual discussions. I would also like to thank Dr. Junpei Kawamoto, Dr. Sabyasachi Dutta, and Nakano-san (KDDI Laboratory) for supportive discussions during the thesis revision. I would like to thank Sakurai Lab secretaries and also Misni Harjo Suwito for their kind help during the exam and public hearing.

I would like to give special thanks to my wife, daughter, and son for their supports during my study and writing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security is a classic problem and one of the most important requirements on the computer systems where the computers process and store sensitive data in many organizations. Since the invention and possible implementation of the time-sharing system, where the processes and data in a computer can be accessed by many people, there are many problems and security breaches involving the sensitive data. Security is often defined in terms of CIA: confidentiality, integrity and availability. Confidentiality requirements dictate who can access which sensitive data. The integrity prescribes that the data should not be changed/updated by unauthorized users. The availability defines that the data should be always accessible by its respective authorized users.

With the advances in the network technology, it is possible to outsource the computational and data storage to cloud services owned by Internet companies. The main advantage of this method is the users do not need to maintain their own physical computing and storage platforms. They can rent the computing and storage platforms at a fraction of the cost to maintain their own data centers. The users can also easily increase the capacity of the computing and storage platform without the need to physically buy new hardware. They can just change their plan to a more expensive one. However, the cloud computing model raise many security concerns because it has a fundamental weakness: the computing and storage platforms are not controlled by the users, so that the users should consider to implements additional security mechanisms to protect the confidentiality and integrity of their sensitive data.

To provide the computing and storage services to their users, a typical cloud systems uses a powerful distributed system (i.e., a grid system) to provide the services. In a distributed system, because the tasks can be executed by many computers, some curious parties (i.e., the auditors) may want to trace the origin and processes that produce the data (i.e., the provenance of data), so that they can judge the integrity and value of the data produced in the systems. The problem is the typical database services normally do not record the information about the origin and processes that produce the data, so that the cloud systems need to record the information using a provenance recording system. The provenance recording system, that is inspired from the provenance in the works of art, is suggested by many researchers to be implemented in a distributed system so that the auditors can completely trace the data history. The provenance recording systems keep the collection of metadata created by the process executors that describe the origins and the processes to produce the data. The metadata should be stored in a special database, we call the Provenance Store, that is accessible to the auditors. The problem is, the malicious parties, rather than attacking the databases, may also try to attack the Provenance Store to change the data history or access sensitive provenance. A trusted provenance recording system should implement security mechanisms to protect the data history and help the auditor correctly judge the value of data.

In this thesis, we are focusing on the problems to protect the integrity and confidentiality of data by implementing secure provenance and applying cryptographic schemes. In computing systems, a security problem can be defined by a description of the state of computer systems and data when the attackers successfully break the systems or data security. The basic methods to protect the computing resources are by implementing access control schemes. Access control prevents unauthorized accesses by employing a trusted reference monitor (that can be implemented by a small program run in a tamper-resistant hardware) that intercepts all accesses and decides what are allowed and what are prohibited based on a security policy. The reference monitor can also be implemented in an operating system to mediate all accesses of users to the computer resources (application, memory or data), in a network router that mediates access to the network resources, or in a database interface that mediates access to data. A main challenge in an access control system is how to define policies that reject all possible malicious accesses while providing minimum accesses that are needed by authorized users efficiently.

Cryptography uses mathematical and statistical properties of cryptographic functions to ensure some security requirements. A central concept in cryptography is the one-way function that is a function that can be easily computed, but it is difficult to compute the inverse. The one-way function can be implemented by some mathematical concepts that were treated pure theoretical (i.e., number theory, abstract algebra). It can also be implemented by a function that is heuristically developed and showed to be resistant to some specific security attacks. The main cryptographic schemes are encryption and digital signature that can be used to protect the confidentiality and integrity of data.

1.1 Summary of the Contributions

1.1.1 An Integrity Scheme for the Provenance Recording System

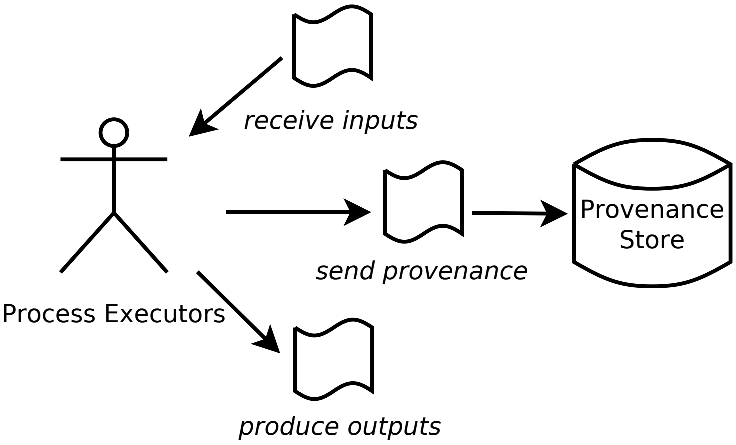

FIGURE 1.1: Provenance recording

In the context of the computer systems, the provenance is recorded in the form of assertions created by the process executors that explain about how to produce the data outputs. The provenance assertions are submitted by the process executors to a dedicated database (in this thesis we call the provenance database as a *Provenance Store* – see Figure 1.1) for a long term storage and easy access by the auditors. The auditors are the parties who need to check and evaluate the processes to produce the data. An assertion submitted by a process executor confirms the responsibility of the process executor to the process and data output.

The assertion can also be an evidence to support the auditor's appraisal about the quality and value of the data output.

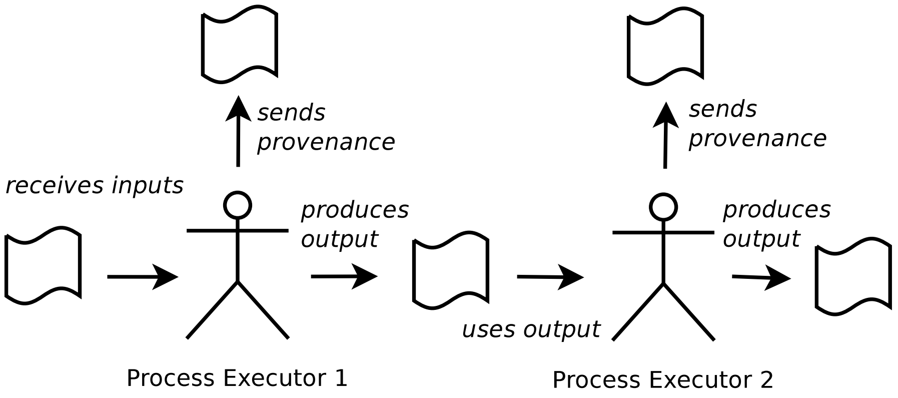

Figure 1.2: Provenance as a proof of responsibility of each process executor

The provenance proves two main facts: the process to produce the data, and the source of the data. In a distributed system, the data can be produced by collaborations of many process executors. The provenance of the distributed processes shows the contribution and responsibility of each process executor to each data output. By checking the provenance, the auditor should be able to trace whether there are more than one process executors should be responsible to the data output. For example, in Figure 1.2, *Process Executor 2* uses the output of the *Process Executor 1*, so that in this case, the output of the *Process Executor 1* affects the output of the *Process Executor 2*. Because the *Process Executor 1* is responsible to the data used by *Process Executor 2*, the *Process Executor 1* should also be responsible in part to the data output produced by *Process Executor 2*. The auditor should be able to confirm the chain of responsibilities by inspecting the provenance and data.

The problem is, the provenance can also prove that some process executors are responsible for faulty processes that produce erroneous outputs. The faulty processes and outputs may cause disadvantages to the process executors who are responsible to the processes and outputs, because the parties who are affected by the outputs of the faulty processes may send complaints to the process executors. The process executors may got reward (be respected) for high quality outputs, but they also may get disadvantages (penalty) if they produce low quality outputs. The honest process executors will take responsibility to the faulty processes and all of the consequences (including bad reputation), and try to improve their credibility later. However, the malicious ones may try to avoid the responsibility by trying to update or delete the provenance of the faulty processes.

In this thesis, we propose an integrity scheme for the provenance recording system. We define the security model as "inconsistent claims" and "inconsistent interpretation" attacks. Our method uses the signature chain method and also uses a Trusted Counter Server (TCS) to label each provenance assertion with a consecutive and unique counter, and stores the latest counter in a trusted storage. We show that the method can detect the "inconsistent claims" and "inconsistent interpretations" problems in the provenance system that cannot be detected by the normal hash/signature chain.

1.1.2 An Access Control Model for the Provenance Recording System

The provenance describes the processes to produce the data. To fully audit the processes, an auditor needs to access all of assertions created by the process executors that have causal relationships with the data. A causal relationship means that a process has contribution or affects the output of another process. The causal relationship is transitive, so that if process A contributes the output of process B, and process B contributes to the output of process C, then the process A should have an indirect contribution to the output of C.

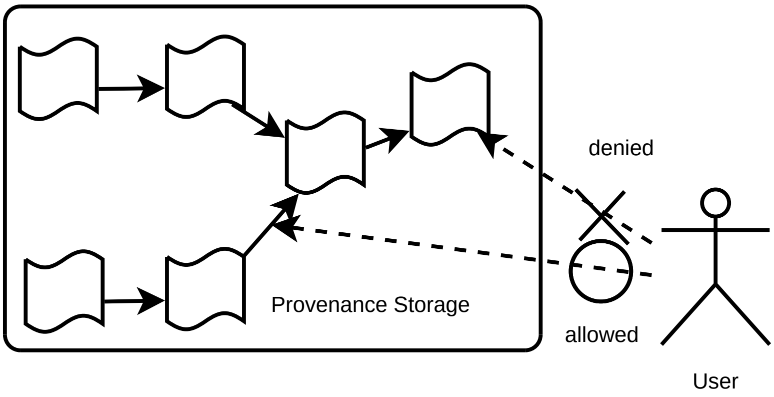

Figure 1.3: Access Control to Provenance

When an auditor needs to check the processes that led to a data object, for a full traceability of the causal relationships, the auditor needs to have access to all connected assertions. However, some of the assertions may be sensitive so that we need to restrict accesses by some auditors. A simple method of access control is by restricting access to each element of the provenance assertion: process documentation and relationship between process documentations (see Figure 1.3). It is desirable to have an access control system that considers the causal relationships in the provenance and supports access control policy efficiently.

We propose an access control method to the provenance that supports access restriction based on the relationships between the provenance assertions. Our access control method restricts the traceability of the provenance assertion using the access right we call TRACE. We combine the access right TRACE with the Multilabels method for better access granularity. We show and evaluate the implementation of our access control method by using a trusted and idealized reference monitor. We also propose an alternative implementation by using an encryption-based access control system for better security.

1.1.3 A Method to Sign a Sequence of Digital Documents

In a part of this thesis, we are concerned about the method to sign a sequence of digital documents (created by a party) that consists of many distinct documents that are created sequentially. The signature of the sequence of the digital documents should prove two facts about the documents: the authenticity of each member of the documents, and the order of the documents in the sequence.

We identified two main differences of signing this type of documents with the signature scheme for a single document. The first one is the new members of the sequence can be added later after signing current existing members. The second difference is during verification of the members, we cannot assume that we have access to all documents in the sequence. We can only assume that we have access to the documents that will be verified.

A simple method to sign the sequence of the documents is by simply appending a consecutive counter to each member of the sequence and signing each consecutive member of the sequence with a standard signature. The signatures can be used to authenticate each member and also the order of the members without having access to other members of the sequence. However, this method has some disadvantages where we need to keep and track the unique identification number of each sequence, and also the counter does not represent the “hard-proof” about the order of the members in the sequence.

We propose a signature scheme for a sequence of digital document by extending the standard signature, so that it can be used to generate the signature sequentially and also verifying the order of the members in the sequence. We show a variant of the signature with message recovery originally proposed by Bellare et al. and propose a signature scheme for the sequence of digital documents using a signature with message recovery as the primitive. We describe the security model of the scheme in the form of Extended Existential Unforgeability under Chosen Message Attack (EEUF-CMA) and Order Unforgeability Under Chosen Message Attack (OUF-CMA). We prove the security of the scheme in the random oracle model.

1.1.4 A Proxy Re-encryption Scheme for the Symmetric Key Cryptography

In a database service provider, the users may encrypt their data to protect the data confidentiality. In a typical implementation, the data is encrypted by the data owner before submitting the data, so that any other parties (including the service provider itself) cannot access the data without knowing the decryption keys (see Figure 1.4). The service provider can be an online provider in different places or organizations. The data owner encrypts the data because he/she may not trust the service provider, but he/she needs to use the database service so that he/she can access the data anywhere/anytime and does not need to be burdened with maintaining the database server. The data owner may also need to share the data to the other users. In an encrypted database, the data owner can share the data by simply providing the encryption keys that can be used to decipher the data.

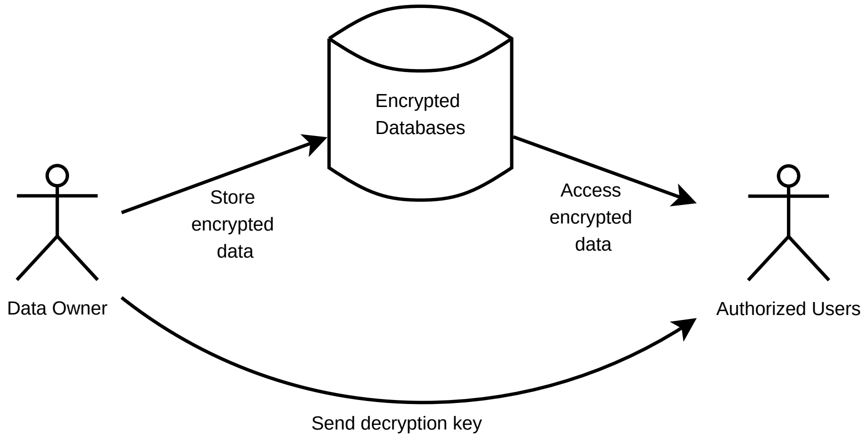

Figure 1.4: An Encrypted Database

A problem in an encrypted database is whenever the data owner needs to update the encryption/decryption keys because of the keys are leaked or the data owner wants to revoke access by the other users. Using the naive method, the data owner needs to download the encrypted data, decrypts the data and re-encrypts with the new key locally and submits the new encrypted data to the service provider. For a large encrypted data, this method is not efficient because the data owner needs to pay computation costs for decrypting and re-encrypting the data and high network cost for downloading and uploading the data.

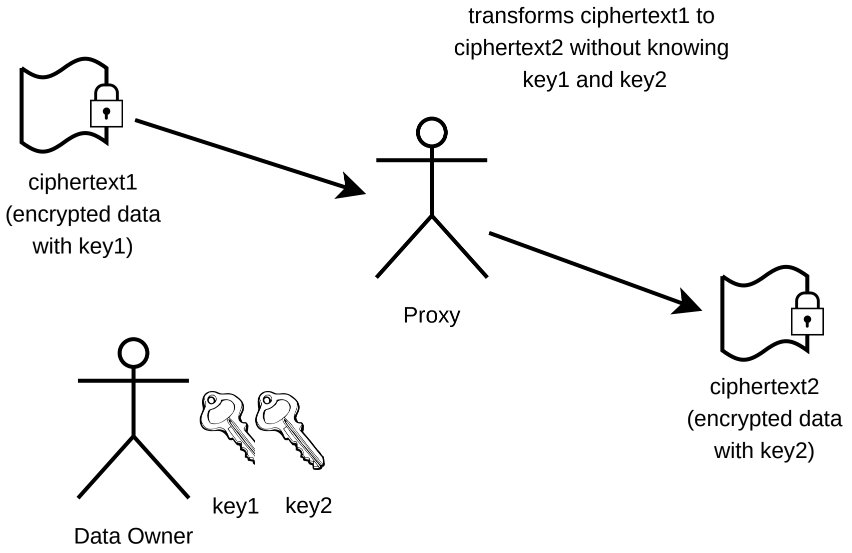
transforms ciphertext1 to ciphertext2 without knowing key1 and key2
ciphertext1 (encrypted data with key1)
Proxy
ciphertext2 (encrypted data with key2)
key1
key2
Data Owner

Figure 1.5: Translating ciphertext encrypted with one key to another without knowing the keys

A more efficient and desirable method is by allowing the service provider to re-encrypt the data with a proxy re-encryption scheme. Using this method, the re-encryption can be delegated to a proxy (that can be implemented in the service provider), without providing any encryption or decryption keys to the proxy. The proxy re-encryption works by finding a function that can directly translate the ciphertext encrypted with a key to another ciphertext encrypted with another key without knowing any encryption/decryption keys (see Figure 1.5).

In the asymmetric ciphers setting, we can use some beautiful mathematical functions and properties (i.e., pairing, homomorphic encryption property) to implement a secure proxy re-encryption [6–10]. However, for performance reasons, the data is normally encrypted with a symmetric cipher (asymmetric encryption is much slower that symmetric encryption).

In this thesis, we propose a secure symmetric encryption scheme that supports fast key update and proxy re-encryption. The idea of our scheme is by first transforming the plaintext using an All or Nothing Transform (AONT) and then exploiting some of AONT's characteristics to implement an efficient and secure proxy re-encryption scheme. We prove the security of the scheme under chosen plaintext attack security model. We also show that the scheme is more efficient than the simple decrypt and encrypt method.

1.2 Thesis Organization

This thesis consists of seven chapters.

- In the first chapter, we discuss the motivation and summarize all of the contributions that are included in this dissertation.
- In the second chapter, we discuss the background on the clouds and the provenance system and also the related projects on provenance system. We also describe our model for the provenance recording system. This model is used as a basis for the development of the integrity scheme and also the access control method described in the Chapter 5 and 6. In this chapter, we also discuss the background on cryptographic techniques.
- In Chapter 3, we describe our proposed method to protect the integrity of the provenance.
- In Chapter 4, we describe our proposal for the access control system that can be applied to a provenance recording system.
- In Chapter 5, we describe our proposed signature scheme that can be used to sign a sequence of digital documents.
- In Chapter 6, we describe our proxy re-encryption scheme in the symmetric cryptography setting.
- Chapter 7 is the conclusion of this thesis.

Chapter 2

Background

2.1 Cloud Computing

The term cloud computing is typically referred to as the usage of computing services that are provided by the cloud companies. The customers of the cloud companies can access the computing services by using the networks (i.e., the Internet). The history of the cloud computing can be traced to the concept of time-sharing, where the resources of a computer (i.e., a mainframe) can be concurrently accessed by many users. With the advances of the network technologies, virtualization software, and distributed computing, it is now possible to provide many types of computing services in the Internet.

Foster et al. define the cloud computing as follows [11]:

> A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.

A cloud computing system is a type of distributed system that is used to provide the computing resources as services [12]. Because the distributed system should provide an interface so that the users believe that they are dealing with a single system, the cloud computing is seen as a centralized system from the user's perspective [12]. From the perspective of the services provided by the clouds, the authors in [13] defines the clouds as everything as services (XaaS). So that, the clouds can include SaaS (Software as a Service), PaaS (Platform as a Service), HaaS (Hardware as a Service), DaaS (Database/Desktop/Development as a Service), IaaS (Infrastructure as a Service), BaaS (Business as a Service), etc. They categorize the cloud services into four layers of the services [13] (from the lowest layer to the highest one): Hardware-as-a-Service, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS).

The authors in [14] argue that the cloud should include both of the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. So that, all layers in the clouds are integral parts that support each other. From the economics perspective, they identify that the clouds provide many advantages to the users/companies, because the users/companies can reduce the cost of maintaining their own hardware, infrastructure, platform or software. Another advantage of the cloud is the scalability. The users/companies can easily increase the computation power or storage to fulfill their business requirements. However, they also suggest that the users should considers the security and reliability of the cloud services.

The authors in [15] found many security problems in the cloud, including the data integrity and data confidentiality. The main data security problems for an enterprise in the clouds are caused by the fact that data are stored outside of the enterprise boundaries. Without encrypting the data, the users have no choice other than trusting the cloud companies to protect their data.

Foster et al. argue that the cloud computing uses many concepts and services provided by the grid computing [11]. The grid was originally proposed as a distributed computing infrastructure for advanced science and engineering [16, 17]. The grid computing is basically a form of distributed computing that provide many services for the resources managements, for example the computing and data resources, virtualization and also the protocols for managements and communications.

Foster et al. state that an important feature in the grid that can also be useful in the cloud is the provenance system that is used to record the history of data [11]. In computer and grid systems, the provenance concept has been applied to record the history of data in the computer systems [18–21]. Other useful applications of the provenance concept are in the health-care applications [3, 22–24], and in the courts and police/law institutions [25, 26].

2.2 Provenance Recording Systems

2.2.1 What is the Provenance?

In the dictionary, provenance is defined as "the place of origin or earliest known history of something" and also "a record of ownership of a work of art or an antique, used as a guide to authenticity or quality" [27]. The word *provenance* in the English vocabulary was imported from the French word *provenir*, so we may say the provenance/origin of the English word *provenance* is the French word *provenir*. Provenance of an object is a very useful information to understand the characteristics of the object. If we know the history about how the French word *provenir* was adapted to the English vocabulary, we may say that we know the provenance of the English word *provenance*. Furthermore, if we know the history of the French word *provenir*, for example its origin or the first person who used the word in conversation or writing, we have better understanding to all words that are derived from it, including its English word version.

The provenance has its root in the work of art where the provenance documents the history of an art object [28–30]. In the work of art, the provenance is used to estimate the quality (and also the value) of the art objects. An object originated from famous artists and owned/maintained by trusted people/organizations has higher value than the object comes from unknown artists or maintained by untrusted people/organizations. In the traditional paper-based world, the provenance is recorded as a collection of documents that describe the origin of the object, and all events related to the object that affect the object's current condition (i.e., who are the owners/maintainers).

In the context of the computer systems, Moreau and other researchers define the provenance as follows:

Definition 2.1 (Provenance as Process)**.** [2, 31] The provenance of a piece of data is the process that led to that piece of data.

This definition says that to record the provenance of a piece of data, we should record the process that led to the data. We may record the process in detail, for example recording step by step of the process execution, or we may record as simple as the name of the process, with an assumption we understand the process from its name. Ideally, the provenance records the detail of the processes, at the level so that we can re-produce the data exactly. However, it is not always practical to record the provenance of the large processes in detail because it needs large documentation storage (it is equivalent to the size of the complete programs executed by the processes). A more practical approach is by recording the higher level assertions about the process execution (i.e., by only recording the descriptions/summaries of the processes, the input parameters and the outputs of the processes).

There are two perspectives of the definition of the provenance in the computer systems [1, 18, 31–35]. The first perspective defines the provenance as the documentation of the detail of data derivation in a database (*fine-grained provenance*). In this perspective, we should record each step that are executed to derive data from its source, for example in a database, we should record each query that are executed to produce the data output.

The second perspective defines the provenance as a documentation of processes execution to produce the data (process-oriented or *coarse-grained provenance*). From the coarse-grained provenance we know the sequences of the process executions, which processes produce each data. However, the detail of data derivation in each process cannot be concluded precisely because there is no specific requirement to the format of the process description (that is why it is called a coarse-grained provenance).

In the coarse-grained provenance, the provenance of an object captures the information about the process to produce the object [33, 35, 36] that includes: (1) the **origin/source** of the object, and also **entities that cause** the existence of the object, (2) description about the **process** to produce the object, and (3) the **actor** that executes/controls the processes. For example, in the medical contexts, the provenance of a medical object (i.e., a medical record) should include the sources of the object (for example, medical tests), the description about the process (i.e., reasoning of diagnosis or the treatment), and also the actor that executes the process (i.e., the physician who write the diagnosis or decides the medical treatments).

The provenance can be recorded by the actor that executes the process and can also be recorded by other parties (either manually by people or automatically by computers). On both cases, there should be a proof of the relation between the process with the actor (either by signatures or other proofs).

2.2.2 Fine-grained vs Coarse-grained Provenance

2.2.2.1 Fine-grained Provenance

In the fine-grained provenance, the data derivation is recorded in detail so that by having the provenance it is possible to reproduce the data as the original processes. A fine-grained provenance can be described as why and where provenance [35]. Why provenance records the detail on how to produce the data, that is the process. Where provenance records the origin/source of the data. For example, in a database, why provenance is a collection of queries that were executed to produce the data. Where provenance is a collection of the location of the data sources in the database. The definition of why provenance for relational view can be found in [37].

By having why and where provenance, we can reproduce the data verify the origin of data. An application of why and where provenance is to record the provenance in curated databases where the content of the database is the results of interpretation from other databases (raw data). Because the curators may modify the data to "clean" the data [38], the provenance is useful when we need to trace the sources of the modified data. We can define the provenance in the form a relational calculus (in a relational query language: i.e., SQL) to trace the data derivation [35]. The SQL can describe the process to select the data from its sources. For example, we record the provenance during the data curation process, so that later we can answer the question: for a given database query Q, a database D, and a tuple t in the output of $Q(D)$, which parts of D "contribute" to t? [30]. An example is the provenance is recorded in the form the relational calculus in the following query:

```
SELECT name, telephone
FROM employee
WHERE salary $>$ SELECT AVERAGE salary FROM employee
```

By checking the provenance, for an output (`"JohnClark"`, `12344444`), we understand which tuple in the relation database `employee` that affects the output. In this case, it is called "why" provenance, that explains why we get the output. If we ask where the number `12344444` comes from? We may answer that the number comes from the field `"JohnClark"`. This is called "where" provenance, that explains where the output comes from. If any error to the output (for example

John Clark finds that his telephone number is incorrect), we can easily find the source of an error from “where” provenance. John Clark may conclude the source of the error is the field `"JohnClark"` in the database.

2.2.2.2 Coarse-grained/Workflow-based Provenance

Rather than recording the data derivation in detail, in the coarse-grained provenance, we only need to record a higher level description of the process of the data derivation (i.e., a workflow that describes the execution plan in a distributed system). The coarse grained provenance is normally used to record the provenance of data produced in a distributed system where many processes are involved to produce the data. Each process describes how to produce the data and the sources that are used in the provenance. A recorded provenance is a collection of assertions created by the process executors about how to produce the data and origin of the data.

From the coarse-grained provenance we understand how many processes in the distributed system collaborate to produce the data. The coarse-grained provenance also represents higher level understanding of execution of processes in the distributed system [39]. A simple model of execution is a sequential execution of processes [5, 40, 41]. The provenance captures the execution model in the form of a chain of assertions created by the process executors. Hasan et al. define the provenance record and provenance chain as follows:

Definition 2.2 (Provenance chain)**.** [40] A provenance chain for a given document D is comprised of a time-ordered sequence of provenance records $P_1|P_2...|P_i|...P_n$ of length $n > 0$, where two adjacent entries P_i and P_{i+1} indicate that user u_{i+1} obtained D from the user u_i where a provenance record P for a document involves two components: the ownership entry for a document, and the log of the tasks applied on the document by authorized users.

The chain model represents the sequential execution of processes that produce the objects [5, 40, 41]. In this model, the processes are executed one after another where each process uses the output of the process that is executed before the process. The provenance also takes a form of a chain where each link is the documentation of each process. The links are connected for two consecutive processes.

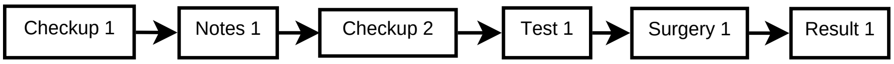

Figure 2.1: Provenance chain of a medical record

Figure 2.1 shows a provenance of a sequential execution of six processes (*Checkup 1*, *Notes 1*, *Checkup 2*, *Test 1*, *Surgery 1*, and *Result 1*).

A more sophisticated model of execution of processes is a directed acyclic graph [18]. For example, this model is used in the provenance of processes in service oriented architecture where the processes are invoked by sending inputs to the services in the systems and the services will send the outputs after each execution. The provenance is recorded by each party after having executed a process or sending data to other parties. Simmhan et al. describe two features of the provenance in distributed system: the origin (ancestral data product(s)) and the process (that transform these ancestral data product(s)) [18]. In the process of audit, the auditor should treat the distributed system as a system that produce the outputs, so the different methods to produce the outputs are represented by the difference sequence/order of the processes execution in the distributed system.

The graph model supports sequential and also parallel execution of processes. The provenance of the processes can also be modeled by a directed graph where a node represents an entity and an edge represents a causal relationship between two entities (i.e., an object is derived from another object, a process uses an entity as its input) [42, 43]. There should be no cycle in the graph because a node in the provenance graph represents the condition of entity at a specific time [44] (in the case of the same process is repeated, the provenance is recorded in a new node). A relationship between two nodes in the graph is a causal relationship, for example a process B uses the output of process A so that the output of A causes the output of B. An annotation can be included in each edge to describe the detail of the causal relationship (i.e., what is the role of an input of a process).

Cohen et al. defines the provenance over a workflow execution as a function which takes as input the identifier of a data object and return the sequence of steps and input data objects on which it depends [45]. All produced data is called calculated data.

Definition 2.3. [45] The provenance of a data object d ($Prov(d)$) is given as:

$$Prov(d) = \begin{cases} (sid, \{d_1 : Prov(d_1), ..., d_n : Prov(d_n)\}) \\ Info(d) \end{cases}$$

The first case is applied whenever d is calculated data, while the second case is applied whenever d is the other data.

For example, a workflow with two processes (S_1, S_2), where S_1 takes as input $\{I_1, I_2\}$, produces as output $\{D\}$, which is taken as input to S_2, which produces as output $\{O_1\}$. Then

$$\begin{aligned} Prov(O_1) = (S_2, \{D : (S_1, \{I_i : Info(I_1), \\ I_2 : Info(I_2)\})\}) \end{aligned}$$

Provenance can also capture the information from the computational tasks. There are two forms of the provenance for computational tasks: *prospective provenance* and *retrospective provenance* [46]. The *prospective provenance* captures the workflow to generate the data product (plan of distributed processes execution) while the *retrospective provenance* is a detail log of the execution that can be used to reconstruct the workflow (documentation during execution of the workflow). An important component of the provenance is causality that represent relationship between entities during the execution of the workflow

2.2.3 Open Provenance Model

An example of the graph model of the provenance is the Open Provenance Model (OPM). The OPM is being proposed by the provenance research community as a standard provenance model [33, 47, 48]. The model is a directed acyclic graph (DAG) with three types of nodes and five types of edges. The types of nodes are: (1) the artifact, that is immutable information (i.e., an input or an output), (2) the process, that is the series of action on or caused by artifacts and resulting in new artifacts, and (3) the agent is the active entity that starts/controls a process (see Figure 2.2). The types of relationships between the nodes are as follows (represented by edges in the graph): (1) an artifact was *used* by a process, (2) an artifact *was generated by* a process, (3) a process *was triggered by* a process, (4) an artifact *was derived from* an artifact, and (5) a process *was controlled by* an agent. The OPM standard does not specify the internal provenance representation and the protocol to store or query the provenance graph to/from a storage [33]. The OPM model also does not specify how to record and secure the provenance.

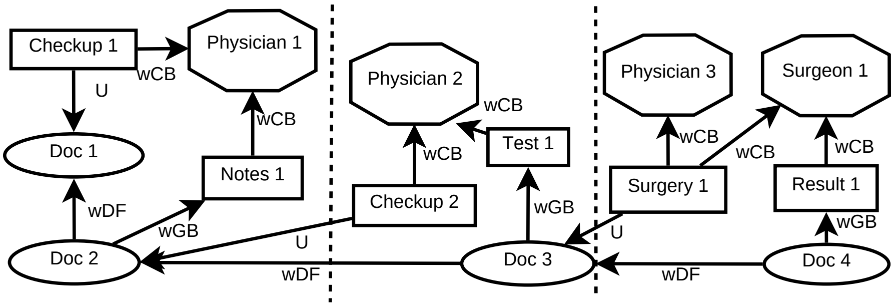

FIGURE 2.2: An example of the Open Provenance Model [1]

(U=Used, wDF=wasDerivedFrom, wGB=wasGeneratedBy, wCB=wasControlledBy)

The OPM can be represented graphically by using an octagon as an agent, a rectangle (or a square) to represent a process and an ellipse (or a circle) to represent an artifact. In Figure 2.2, we show an example of the OPM in the medical context. This example is adopted from the example of the OPM in [1]. In this example, a patient medical records are created by three physicians and one surgeon (*Physician 1, Physician 2, Physician 3*, and *Surgeon 1*). Initially, the *Physician 1* does a checkup (*Checkup 1*) that uses a previous medical record (*Doc 1*). The *Physician 1* writes a note that is recorded in the *Doc 2*. In the next checkup (*Checkup 2*), the patient meets the *Physician 2* who reads the *Doc 2*, does a test (*Test 1*) that produces *Doc 3*. In the third visit a surgery (*Surgery 1*) is done by *Physician 3* and *Surgeon 1*. They write the result (*Result 1*) in the *Doc 4*.

2.2.4 Provenance vs Version Control System

The existing system that is very related to the provenance is the version control system that is also known as the source code control system [49–51]. A version control system maintains multiple versions of the documents and the changes (difference, Δ) between the versions. Typical examples are Concurrent Versions Systems (CVS), and Subversion [52]. A manual of the Subversion [53] describes the main capability of the Subversion as follows.

> Subversion manages files and directories over time. A tree of files is placed into a central repository. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your data, or examine the history of how your data changed. In this regard, many people think of a version control system as a sort of "time machine". [53]

A version control system can be seen as a form of the provenance system because it also records the history of update. It records a specific form of the provenance where we define the process as document editing process. De Nies et al. [54] conclude that the version control system records an aspect of the provenance: entities, activities, and people producing or modifying a piece of data. Koop et al. [55] analyze that the current implementation of the provenance has a weakness, because the provenance cannot provide a strong link between the outputs that can be cured with a version control system. They argue that the provenance and version control are complement each other where the provenance can explain how the data is produced and the version control explains the changes between data. By combining the version control with cryptographic hash, they incorporate the strong links between inputs in the existing provenance system [55].

Cheney et al. [56] analyzed that the version control system, the operating system log and other records to the changes of files are some forms of the provenance. However, they are still not representing a complete provenance. Halpin et al. [57] developed a provenance for the data in RDF (Resource Description Framework) by incorporating versioning system to the RDF's data.

We conclude the main differences of the provenance and version control system as follows.

1. The version control has a strong relationship, where we can check whether the document is derived from another document by checking the differences (Δ).

2. The provenance records any relationships, not only the differences, so that the provenance is more general than the version control. We can define the relationship with a new definition, for example a document A has a relationship with document B if A affects the process to produce document

B, for example A is an input or parameter that is used by the process that produces B.

2.2.5 Related Projects on the Provenance

2.2.5.1 EU Provenance Project

The EU provenance project focuses on the framework to implement the provenance in the context of e-science and healthcare management. E-science is about using computation and data resources to help scientists to get scientific results [20]. In e-science, the scientists are doing *in silico* experiments to analyze the data products by using many services provided by many organizations (internal or external of the organization where the scientist are working). One motivation of e-science is to develop a better collaboration between scientists by sharing computation and data infrastructures. E-science normally uses grid-infrastructure to manage sharing of resources across many organizations.

A scientist performs *in silico* experiment by composing a *workflow* that describes how to combine many computation and data resources to get a result [18]. The scientist who invokes the services may want to verify that the executions were performed correctly or conform to some criteria [21]. The scientists also want to review the result of experiments started by other scientists. In these cases, the provenance can be used to trace and record the processes execution that produce outputs in the e-science experiments. The provenance is an alternative to *static verification* where each program that run in services is verified to conform the criteria or *run-time* checking that verify the program at run time [21].

Provenance can be used to record the history of the execution of processes so that the scientist can check whether the services correctly execute their invocation and the other scientists can verify and audit the sources of the data inputs and the steps executed to produce the results. The information includes the the process (the description of process and identity of the service), data inputs and the parameters used to execute the processes, and the outputs. Figure 2.3 shows the provenance recording and user interfaces architecture for provenance-aware applications proposed by EU Provenance project. The central of the architecture is the Provenance Store where Application Services that execute the processes

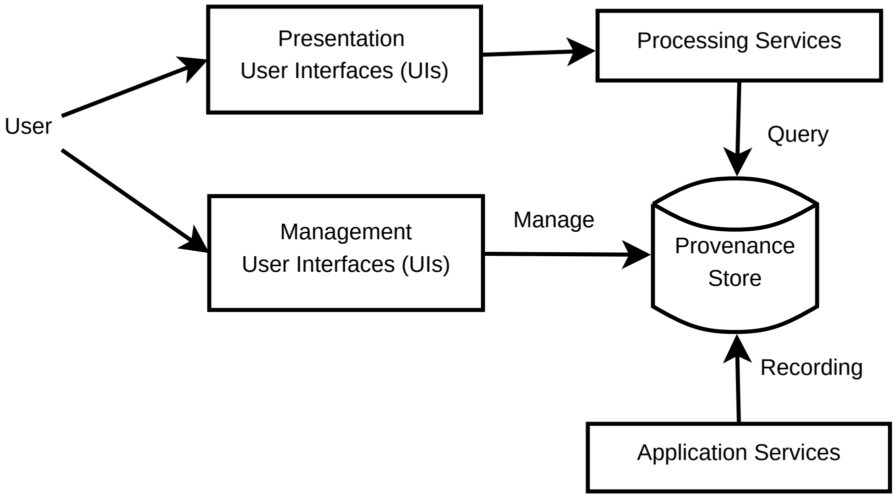

FIGURE 2.3: An Architecture of Provenance-Aware Application Proposed by EU Provenance Project [2]

record the provenance and Processing Services that query the provenance from the Provenance Store to be sent to the user interfaces.

To implement the provenance system, the EU provenance project develop model of provenance record using the concept of *p*-assertion and the Open Provenance Model [2, 33, 47, 48]. They defined *p*-assertion as an assertion that is made by an actor and pertains to a process. The provenance (documentation of a process) consists of a set of *p*-assertions made by the actors involved in the process. What should be recorded in the *p*-assertion is specific to the requirements in each system.

2.2.5.2 Provenance in Healthcare Management

In healthcare management system, we need to share the healthcare data across healthcare boundaries so that we can trace the medical history of a patient for better understanding for the current patient's health condition. Documents in a hospital include the patient medical records that are produced by many physician, and medical laboratory staffs from many different healthcare organizations. Because the organizations may be independent, a standard for information exchange is needed.

A standard framework that can be used is a distributed Electronic Healthcare Record (EHCR) System [3, 22, 58]. The EHCR (see Figure 2.4) provides the system to collect the patient healthcare record from multiple database systems. It uses the ENV13606 standard, for the messages, and communication rules. ENV13606 standard provides three types of messages: (1) request message, (2) notification message, and (3) message that contains privacy protection rules [3].

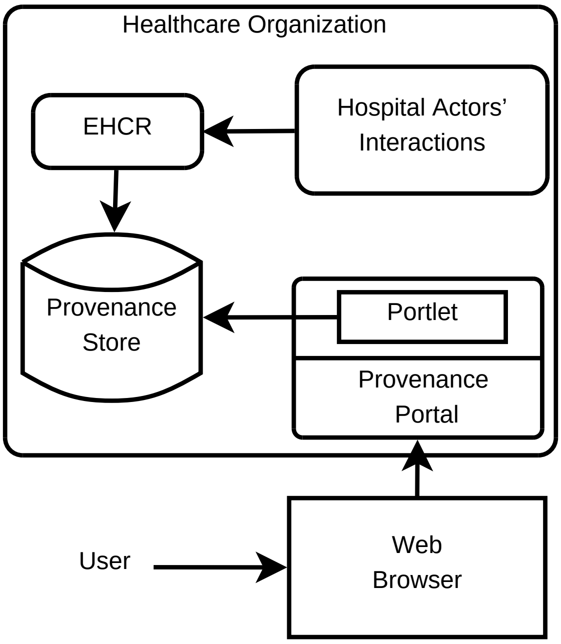

Figure 2.4: An Architecture of Portal-EHCR [3]

In a hospital, the provenance describes the causal relationships between the objects (i.e., medical records, medical test results), the processes that produce the objects (i.e., a medical checkup, medical diagnosis) and the actors that control the processes (i.e., the physicians). Kifor et al. [22]) describe the case of provenance application in an Organ Transplant Management (OTM) application that uses a subsystem of EHCR. The challenging issues in implementing provenance in OTM are [22]): (1) the provenance should record the process execution carried by human being (i.e., physician) (2) the past treatments are relevant to the current treatment while there are not always strong connections in the provenance of previous treatments to the current treatments (3) privacy problem because the agent who manages the provenance knows much about the patient than any other agent.

A physician checks the provenance to know the past history of a patient and to understand the causes the current patient condition [3]. A patient wants to prove a wrong treatment or laboratory test by a doctor or a laboratory staff by showing the provenance. A physician wants to show he/she did not do a mistreatment to a patient by showing the provenance of the treatment [5]. An independent reviewer wants to know the cause of the death of a patient after having a medication or a medical treatment.

2.2.5.3 Provenance Aware Storage System (PASS)

Provenance-aware storage system (PASS) is a provenance implementation in the storage [59]. The main motivation is to support better traceability for processes execution. PASS automatically collects and stores the provenance and provides management interface for the provenance. We can use PASS to search and analyze the provenance, for example comparing two provenance. Figure 2.5 shows the architecture of the PASS. It is implemented in Linux OS.

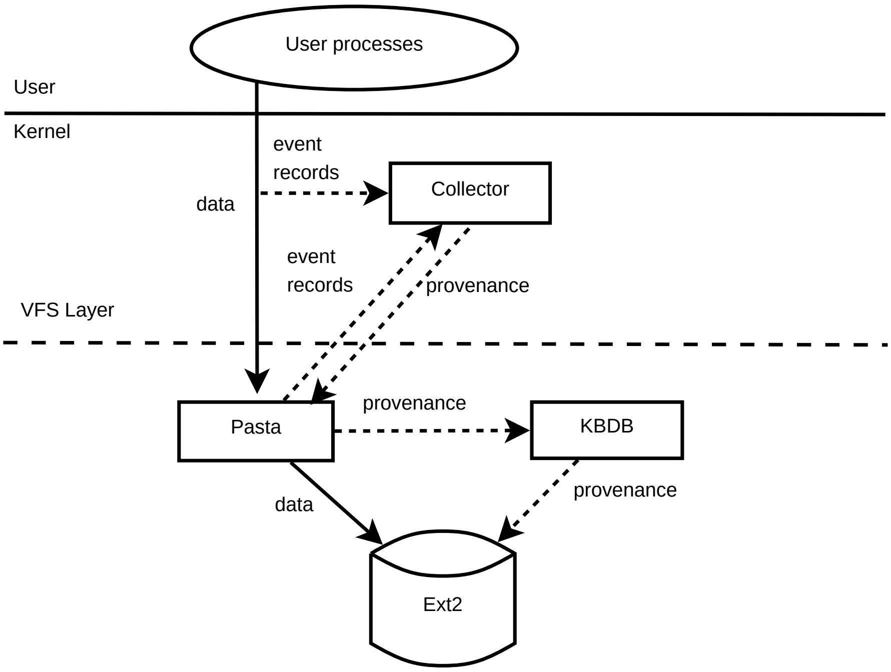

FIGURE 2.5: An Architecture of Provenance-Aware Storage Systems (PASS) [4]

PASS mainly consists of two components: the storage system and the provenance collector. The provenance collector intercepts activities in the system, creates the provenance to be stored in the indexed provenance storage implemented in KBDB (an in-kernel port of BerkeleyDB engine) [4]. PASS uses the query tools in the database engine for searching and analyzing the provenance. PASS records the provenance in the form of a graph where the nodes consists of the processes (for example bash, config), a workflow engine, and the output. The process execution is managed by the workflow engine.

2.2.5.4 Sprov Library

The Sprov Library is developed by Hasan et al. [5]. It is an implementation of the provenance concept in a system similar to the PASS project. Hasan et al. define that a document can be a file, database tuple, or network packet. The provenance of a document is defined as the record of actions taken on the document. Each access to a document D generates a provenance record P. P may include many information, for example the identity of the principal, a log of the actions and their associated data, a description of the environment when the action is executed, and confidentiality and integrity related components, such as cryptographic signatures, checksums, and keying material [5]. A provenance chain for a document D is a time-ordered sequence of provenance records $P_1 \| P_2 \| ... \| P_n$. The chain can be stored by simply appending it to the document D.

To protect the provenance chain, they implement hash/signature chain C as follows:

$$C = S_U(hash(U, W, hash(D), public, I)|C')$$

where U is the user identity and S_U is a signature by U, W is description of the changes to the document D, *public* is the public key of user, and I is an encryption key to implement confidentiality by using broadcast encryption [5].

Sprov Library is implemented as application level library that consists of wrapper functions for the standard I/O library (stdio.h). Sprov captures the I/O operation executed by application, creates new provenance chain when the application write new data (see Figure 2.6). The data and the provenance are stored in the same storage.

2.2.5.5 Panda: A System for Provenance and Data

In Panda [60], provenance (also called “lineage”) captures where data came from, how it was derived, manipulated, and combined, and how it has been updated over time. Some functions of the provenance: (1) explaining how the data is derived, (2) verifying the correctness of the process that produce the data, (3) allowing re-computation of the data that may be produced by erroneous/outdated sources.

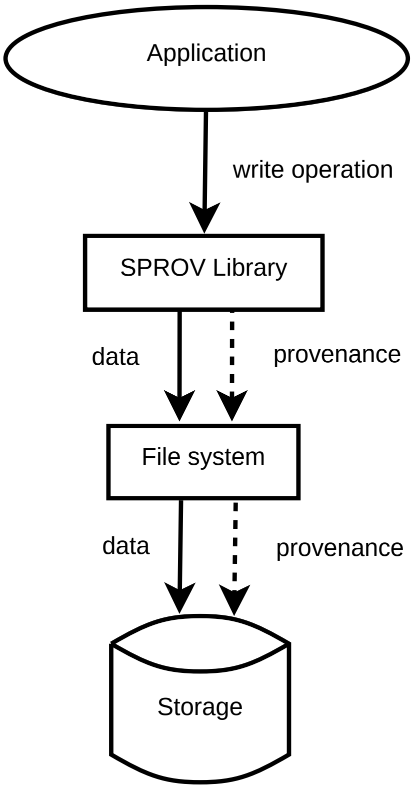

FIGURE 2.6: Provenance recording process in the Sprov Library [5]

The provenance is recorded in two forms [60]:

- **Backward tracing**. Given a data element D, where did D come from? And, what data elements and processing contributed to D?
- **Forward tracing**. Given an input or derived data element D, where did D later go? And, what processing nodes did D subsequently pass through and what data elements were produced?

2.3 Modeling the Provenance Recording System

In this section, we develop a model of the provenance recording system that includes the distributed system model, the storage model, the participants, the process execution, and the provenance recording process. This model is a reference model for the provenance recording system that will be used to describe the security schemes proposed in Chapter 3 and Chapter 4.

2.3.1 Preliminaries

A set is a collection of distinct elements. For example a set $P = \{2, 3, 5, 7\}$ consists of four numbers as its elements. A set of elements of the same type, for instance $\{X_0, X_1, ..., X_{n-1}\}$ is represented by $\{X_i\}$. An element in $\{X_i\}$ is represented by X_i. We can use a more complex representation of the set of element with the same type, for example we can represent a set $\{\langle X, Z_0\rangle, \langle X, Z_1\rangle, ..., \langle X, Z_{n-1}\rangle\}$ as $\{X, Z_i\}$, and a set $\{\langle X_0, Z_0\rangle, \langle X_1, Z_1\rangle, ..., \langle X_{n-1}, Z_{n-1}\rangle\}$ is represented as $\{X_i, Z_i\}$. We use the term tuple to represent a collection of data or variables. A tuple with three elements a, b, c is represented by $\langle a, b, c\rangle$. A tuple can have subtuples, for example a tuple $\langle a, \langle b, c\rangle\rangle$.

We use some variables to represent data items and data in tuples that are sent through network or stored in a database. We also use some functions that take some inputs (variables or numbers) and produce outputs that is represented by $\mathsf{Function}_{identifier}(Inputs)$. The variables and functions (including their identifiers) are described each time they are first introduced. For example, in Section 2.5, we introduce variables $PAsrt$, A, Cid, I, O, Pid, and Pid'. In some parts of this thesis, we introduce functions Hash, Sign, Enc, and the other functions.

We use $\mathsf{ref}(Y)$ to represent a unique reference to a data represented by variable Y so that we can retrieve the data Y from a database by providing its reference. In implementation, reference can be implemented by as simple the name of file/record in the database, or by a Uniform Resource Identifier (URI) that can be used to identify the data universally.

Communications and queries between two parties where the party A sends data or a query to the party B through network are represented as $A \rightarrow B : Data$ and $A \rightarrow B : \mathsf{Query}(Inputs)$. For example, A sends data X to B is represented by $A \rightarrow B : X$. A sends query $Store$ with inputs X to B is represented by $A \rightarrow B : \mathsf{Store}(X)$.

2.3.2 Modeling the Distributed System

In our model, the provenance system records the sources and processes that contribute to the data in a distributed system. A distributed system is defined as [12, 61]:

> A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

The computing elements (also called "node"), can be either hardware device or software process [12]. In this thesis, we call the computing elements as "process". The important element of the distributed system is that the users believe that they are dealing with a single system (it is a centralized system from the user's perspective), so that there should be a method of collaboration between the procesess [12].

We model the collaboration between processes as a centralized execution of an Execution Plan (for example a *workflow* in a grid system) by an Execution Manager. The centralized execution model of the distributed process execution [62, 63] assumes an entity who starts and manages the processes execution, in this model the entity is the Execution Manager. The Execution Manager executes an Execution Plan that is defined as follows.

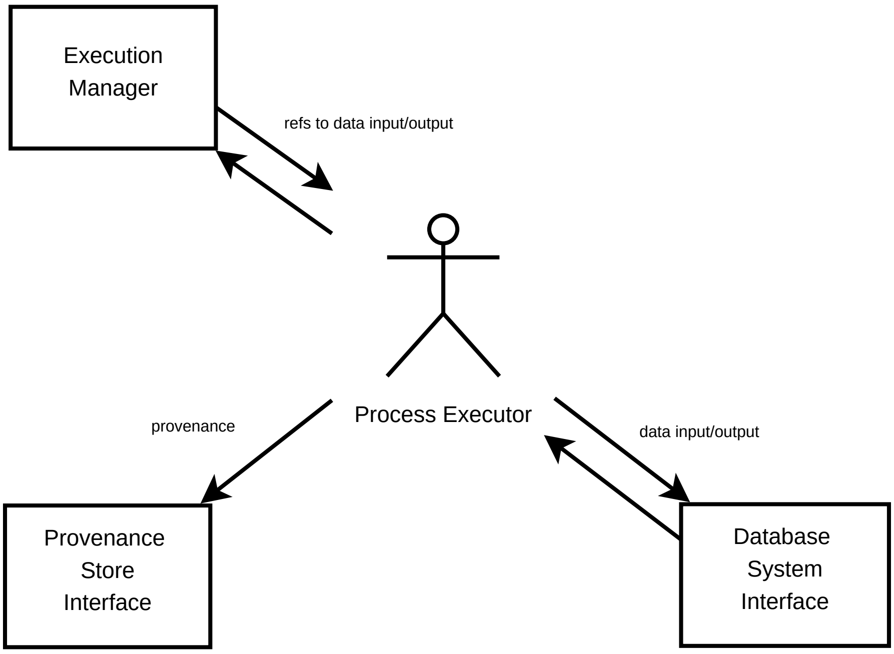
Execution Manager
refs to data input/output
Process Executor
provenance
data input/output
Provenance Store Interface
Database System Interface

FIGURE 2.7: Execution Manager

Definition 2.4. Execution Plan EP for a data set D stored in a database DB is a set of execution nodes $Q = \{Q_0, Q_1, ..., Q_{m-1}\}$ for $m > 0$ and a binary relation F on Q where each execution node Q_i consists of an identification Qid, a process executor Cid, and a list of references to a set of inputs I for $I \in D$. The relation F represents the execution edges such that for $(Q_x, Q_y) \in Q \times Q$ and $x \neq y$ and the execution node Q_y takes the output of execution node Q_x as its input.

The Execution Manager executes the Execution Plan EP by sending the references to inputs to each process executor listed in the execution nodes defined in the Execution Plan. The Execution Manager is responsible to manage the execution so that the relationships between execution nodes (the execution edges) are fulfilled. At first, the DB only stores the data before the execution. Any execution nodes that use the outputs of the other execution nodes that are not yet executed (so that the list of references to inputs are not yet available) cannot be started before all of the inputs are available. The Execution Manager should update the list of references to inputs that are available after a process execution. The Execution Plan can be dynamic, so that the Execution Manager can add new nodes and edges. However, the nodes that have been executed and all edges that connect the nodes that have been executed cannot be removed/deleted.

2.3.3 Modeling the Storage

The access to the storage by each process is needed for data sharing. A simple model is a centralized storage [12, 55] where the data and provenance resides in a storage that is accessible to all processes. The centralized storage simplifies the data sharing because each process can access and use exactly the same data in the same storage. Another choice is distributed storage [12, 55] where the data is shared using the peer-to-peer network (like BitTorrent [64], and also Blockchain [65]). In this model, each process keeps their own storage and advertises their storage contents to be synced or accessed by other processes. In our model, we use the simple centralized storage for the data and the provenance.

Main provenance systems use the concept of the Provenance Store [2, 42, 66], that is a system that has interface to store and query provenance record (showed in the Figure 2.8). The Provenance Store normally provides the interfaces for provenance recording, provenance query interface and provenance management. This architecture is much similar to the database system where a user can query the data in the database.

The Provenance Store PS is the database where the provenance is submitted for a long term storage. Provenance Store Interface provides the interfaces for recording, querying and managing provenance in the Provenance Store. The Provenance Store Interface is also a server that stores the semantic of provenance that can be

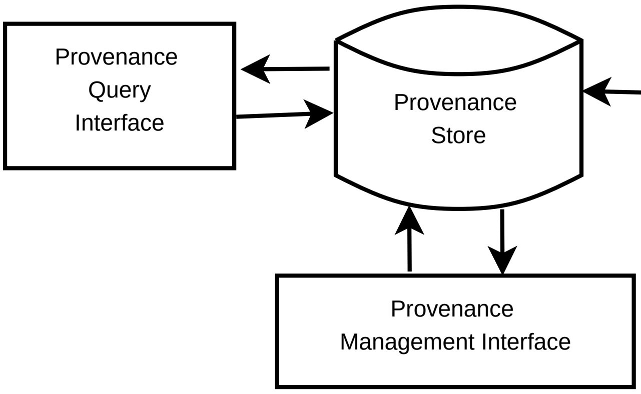
Provenance Query Interface
Provenance Store
Provenance Recording Interface
Provenance Management Interface

Figure 2.8: Provenance Store

accessed by any parties in the system for a common understanding of the meaning of the provenance.

There are three choices of storage of the provenance and data [34]: (1) no separation of the storage of data and provenance, (2) the data and the provenance are logically separated in the same physical storage, and (3) the data and the provenance are physically separated. The choice of the storage affects the way to link the provenance and the data. The provenance system should have addressing and linking mechanisms that are used in the mapping between the provenance and data it is documented, so that from the references to data (inputs and outputs) recorded in the provenance nodes, the auditor knows the location of the data.

The easiest method of the linking is in the first choice of the storage, because we do not need to specify the place (i.e., IP address) of the data and the provenance, they reside in the same storage. In the second and the third storage models, we need to have a linking mechanism that connect data in different storage (logical or physical), so the address of the data or provenance should include the address of the data storage. However, a separate provenance storage is convenient for recording provenance in distributed processes (i.e., service oriented architecture) because it has advantages in accessibility and scalability [2]. In a separated storage there should be a naming and addressing convention to refer to a data location in other places/servers. In our model, we use the third choice where the data and provenance are stored in different physical databases because it is more general and can be applied in many systems.

2.3.4 Modeling the Parties

We identify the parties that are involved in the provenance recording system are as follows (see Figure 2.9):

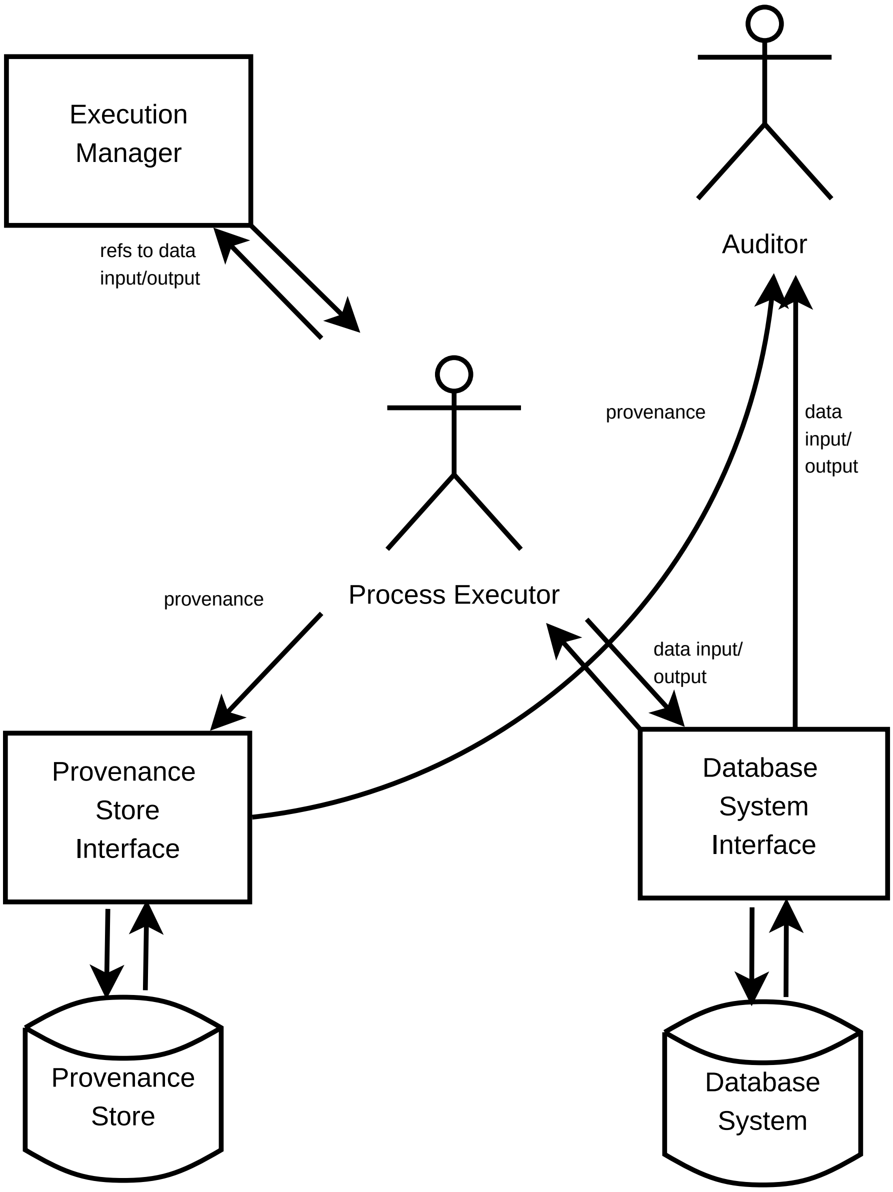

FIGURE 2.9: A Model of Provenance System

1. The Process Executors

   We need to define the process execution in the distributed processes. The distributed system is consisted of a set of asynchronous processes that do not share a global memory and clock. The message transfer are also asynchronous and we assume that each process is running on different processor and the execution of each process is sequential. The Process Executors are the the entities (i.e., computers/services) that receive the inputs from the Execution Manager, execute the processes to produce the outputs, and send the outputs to the Execution Manager.

2. The Database System (DB) and Database System Interface (DBI)

   The Database System is the storage for data inputs and outputs of processes execution in the system. The Database System Interface is an interface to the Database System.

3. The Provenance Store (PS) and Provenance Store Interface (PSI)

   The Provenance Store is a persisten storage where the provenance is recorded for a long term provenance management. The Provenance Store Interface provides an interface to access the provenance in the Provenance Store.

4. The Execution Manager

   The Execution Manager is an entity that starts the execution of processes and stores the Execution Plan. The Execution Manager starts a process by querying inputs from the Database System, sending the inputs to the Process Executors, receives the outputs from the Process Executor and stores the outputs to the Database System.

5. The Auditors (ADT)

   The Auditors are entities that audit the provenance in Provenance Store. The Auditors need to verify the quality of outputs from the provenance or finding flaws in the process executions.

2.3.5 Our Definition of Provenance

In this thesis, we define the provenance as a coarse-grained provenance, formally:

Definition 2.5 (Provenance definition in this thesis)**.** Provenance related to the data set $D = \{D_0, D_1, ..., D_{m-1}\}$ for $m > 0$ stored in a database DB is a set of provenance assertions $P = \{P_0, P_1, ..., P_{n-1}\}$ for $n > 0$ recorded in a database PS. A provenance assertion P_i is a documentation of a process execution at specific time that consists of a process documentation a_i and relationship documentation r_i, where a_i consists of at least an identification number Pid, a process description

A, an identity of the process executor Cid, the list of references to a set of inputs $\{\mathsf{ref}(I_i)\}$, for $I \subseteq D$ and a reference to an output $\mathsf{ref}(O)$ for $O \in D$ and r_i consists of at least identities of the process executors $\{Cid'_i\}$ and the identification numbers of the provenance assertion for the processes that produce $\{\mathsf{ref}(I_i)\}$, that is $\{Pid'_i\}$.

The process description A is a documentation that describes the steps that are executed in the process to produce the output O from the collection of inputs I. A can be as simple as the process name and also a detail program execution. The process executor Cid is the identity of the actor that executes or be responsible to the process. The actor can be a computer or a service and can also be a human being. In this definition, we restrict each process to only have one output and one process executor. In implementation, the process with more than one outputs can use a collection mechanism to collect all outputs into one entity that represents the outputs.

Based on Definition 2.5, a provenance of process that takes a collection of inputs $\{I_i\}$, produces an output O, executed by process executor identified by Cid is stored in a database PS in the forms of $PAsrt$ as follows:

$$\begin{aligned} PAsrt =& a_i | r_i \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \end{aligned}$$

where:
Cid = the ID of the process executor
Pid = the ID of the provenance node
A = assertion about process execution
$\mathsf{ref}(I_i)$ = a reference to an input of the process
$\mathsf{ref}(O)$ = a reference to the output of the process
Cid'_i = the ID of the process executor that produce the input $\mathsf{ref}(I_i)$
Pid'_i = the ID of the provenance of the process that produce the input $\mathsf{ref}(I_i)$

2.3.6 Provenance Graph Model

The provenance in Definition 2.5 can be modeled by a directed acyclic graph (DAG) as depicted in Figure 2.10. We call the model as the uniform DAG model. In the uniform DAG model, a provenance node represents a documentation of a computational entity that consists of a description of process A, a list of process executors C, a list references to the inputs I, and a list of references to the outputs O. An edge that connects the first node to the second node represents a relationship between the computational entities where the computational entity documented by the second node used the output of the computational entity documented by the first node. We call the model as the uniform DAG model because each node and edge has only one type.

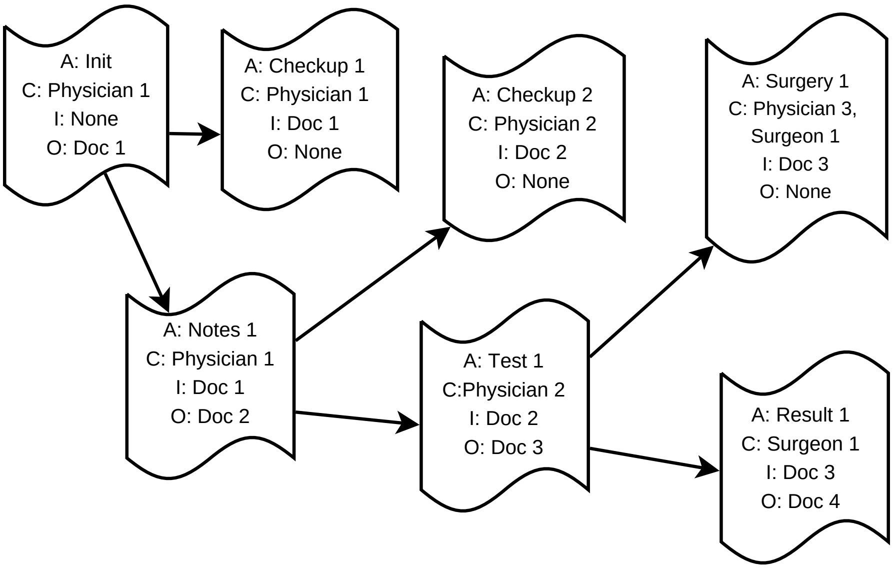

Figure 2.10: The Uniform DAG model

Although it takes the same DAG form as the Open Provenance Model (OPM), the uniform DAG model is different to the OPM model in that it only has a common type of node and a type of edge, while the OPM has three types of nodes and five types of edges as described in Section 2.2.3. Another difference is, in the OPM model there are no edges between an agent with an artifact. To know who are responsible for an output artifact, we should trace the causal relationship from an artifact to a process and from the process to an agent. In the uniform DAG model, the artifact, the process, and the agent (process executor) are collected into a provenance node.

Algorithm 1: Converting the OPM model to the uniform DAG model

```
Input: an OPM graph
Output: the uniform DAG model
for each OPM node where the type is process do
    Create a node, where
    $A \leftarrow$ the OPM process
    $Cid \leftarrow$ the ID to agents connected with “was controlled by”
    $\{\mathsf{ref}(I_i)\} \leftarrow$ references to artifacts connected with “used”, and references to
    artifacts connected to output $O$ with “was derived by”
    $\mathsf{ref}(O) \leftarrow$ reference to a collection of artifacts connected with “was generated
    by”
end for
for each OPM artifact with no “was generated by” connection do
    Create a node, where
    $A \leftarrow$ “Init”
    $Cid \leftarrow$ the ID of agent of process that first uses the OPM artifact
    $\{\mathsf{ref}(I_i)\} \leftarrow$ references to artifacts connected by “was derived from”
    $\mathsf{ref}(O) \leftarrow$ reference to the OPM artifact
end for
return the DAG nodes
```

A node in the uniform DAG model covers all types of the nodes in the OPM: processes, agents, and input/output artifacts. It also represents four causal relationship between the process, artifacts and agents: (1) the outputs (O) “was generated by” the process (A), (2) the process (A) “was controlled by” the process executors (C), (3) the inputs (I) are “used” by the process (A), (4) the outputs (O) “was derived from” the inputs (I). The OPM model can be converted to the uniform DAG model by using Algorithm 1. Figure 2.10 shows the result of conversion of the OPM model shown in Figure 2.2.

Another difference of the uniform DAG model with the OPM is it does not support an inessential feature of the OPM, that is *account*. An *account* is a different detail of view of the provenance [33, 47, 48]. The *account* is useful to simplify the presentation of a provenance graph by omitting some nodes and hiding some details (however, there should be an *account* that record all of the details). A relationship in the OPM, that is “was triggered by” relationship, uses this feature. A “was triggered by” relationship represents a relationship between two processes, i.e., process A and process B, where the process B used the output of the process A without explicitly defined the output of the process A. This relationship exists in an *account* view that represents a less detail process execution where an artifact (that is output of A which is also input of B) is removed from the view. In the the uniform DAG model, all the outputs and inputs of a process are clearly stated. No feature to group some nodes for simpler/higher level presentation.

2.3.7 The Provenance Recording Protocol

The provenance should be recorded to the Provenance Store by parties in the system. Groth et al. describe a provenance recording method where all entities who are involved in the process execution submitted the provenance. [2, 32, 66, 67]. In their model, the provenance is submitted by all parties who are involved in the process execution. For example, when a client invokes a service in the system by sending the inputs to the service, the provenance of invocation is recorded by both client and service that send and receive the inputs. The Provenance Aware Storage System (PASS) records the provenance automatically in an operating system as a sequence of system calls used by a process in the process execution [59].

In our model, we consider the case of the provenance recording method, where the provenance is only recorded by the process executor. Our rationale is because to analyze the security, we need to reduce the assumption about the secure parties. If the provenance is recorded by other parties (i.e., the workflow manager), we need to assume that the workflow manager is trusted, otherwise we cannot consider the provenance submitted by the parties as correct. Assuming the workflow manager as a trusted party is a strong assumption that cannot easily be guaranteed in an untrusted distributed environment.

We define the provenance recording protocol as follows:

1. Process Invocation
   the Execution Manager sends command to execute the process by providing the identification number of the Execution Plan Qid, the references to inputs $\{\mathsf{ref}(I_i)\}$, and the provenance of inputs.

$$EM \rightarrow C \quad : \quad \mathsf{Execute}(Qid, \{\mathsf{ref}(I_i)\}, \{Pid'_i\})$$

2. Process Execution
   The Process Executor retrieves the inputs from DB through its interface

DBI. The Process Executor executes the process, stores the output O to the DB and sends back the reference of the output ($\mathsf{ref}(O)$) to the Execution Manager.

$$\begin{aligned}
C \rightarrow PSI \rightarrow PS &: \mathsf{Check}(\{\mathsf{ref}(I_i)\}, \{Pid'_i\}) \\
PS \rightarrow PSI \rightarrow C &: \mathsf{true}|\mathsf{false} \\
C \rightarrow DBI \rightarrow DB &: \{\mathsf{ref}(I_i)\} \\
DB \rightarrow DBI \rightarrow C &: \{I_i\} \\
C \rightarrow DBI \rightarrow DB &: \mathsf{ref}(O), O \\
DB \rightarrow DBI \rightarrow C &: \mathsf{success}|\mathsf{fail}
\end{aligned}$$

3. Provenance Recording
   The Process Executor creates the provenance assertion $PAsrt$

$$\begin{aligned}
PAsrt =& a_i|r_i \\
a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\
r_i =& \{Cid'_i, Pid'_i\}
\end{aligned}$$

and sends to PSI. The Process Executor reports to the Execution Manager whether the whole process is successful or not.

$$\begin{aligned}
C \rightarrow PSI \rightarrow PS &: \mathsf{SubmitPAsrt}(PAsrt) \\
PS \rightarrow PSI \rightarrow C &: \mathsf{success}|\mathsf{fail} \\
C \rightarrow EM &: \mathsf{Report}(Qid, \mathsf{ref}(O), Cid, Pid, \mathsf{success}|\mathsf{fail})
\end{aligned}$$

2.4 Basic Cryptography

The term cryptography comes from the Greek words *kriptos* (meaning “hidden, secret, concealed”) and *graphein* (meaning “to write”). So that, literally, cryptography means “to write a hidden/secret message”. In the Oxford Dictionary, cryptography is defined as “the art of writing or solving codes” [27]. Historically, cryptography was used for secret writing or secret communications to secure communications between parties, while in modern usage cryptography is used for other security purposes, for example authentication, digital signature, key exchanges protocol, electronic auctions and elections, and digital cash [68].

In the classical cryptosystems, cryptography only deals with private key encryption setting where two parties who need to communicate securely share a secret key. To send the message securely, the sender encrypts the message by transforming the message with a secret key into an unintelligent form (we call as ciphertext). The receiver who knows the secret key can decrypt the ciphertext to the original message.

The modern cryptography includes both of the private key and the public key cryptosystems. The public key encryption techniques that require no sharing of secret keys prior to encryption (but rather require publishing the public keys) are among modern cryptographic techniques and also fundamental inventions in computer science.

In this chapter, we review the basic techniques used in cryptography. The techniques include the collision resistant hash functions, the private key encryption, the public key encryption, and the digital signatures. We also describe the methods to prove the security of the cryptosystems.

2.4.1 The Primitives

2.4.1.1 Collision Resistant Hash Functions

The basic primitive of the cryptographic scheme is the collision resistant hash function. The collision resistant hash function is a form of one-way function that also has collision resistant property. A one-way function is a function that can be easily computed, but it is difficult to compute the inverse. A definition of the one-way function (the strong one), can be found in [69]:

Definition 2.6 (Strong One-Way Function [69])**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is called (strongly) one way if the following two conditions hold:

1. Easy to compute: There exists a (deterministic) polynomial-time algorithm A such that on input x algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$).

2. Hard to invert: For every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's,

$$\mathsf{Pr}[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

The following function is believed to be a one way function if we assume the problem to invert the discrete logarithm problem is a difficult/hard problem [68].

$$f_p(x) = g^x \mod p \text{ for any large prime p}$$

A (cryptographic) hash function is a type of the one-way function with additional property: collision resistant, as defined in [70].

Definition 2.7 (Collision Resistant)**.** Let $H : K \times M \to Y$ be a hash function, the advantage of an adversary B to find the collision of the outputs of H is defined as follows:

$$\begin{aligned}\mathbf{Adv}_H^{coll}(B) \quad &= \Pr[K_i \overset{\$}{\leftarrow} K; (M, M') \overset{\$}{\leftarrow} B(K_i) : \\ &(M \neq M') \wedge (H_{K_i}(M) = H_{K_i}(M'))]\end{aligned}$$

For a secure cryptographic hash function, the advantages of the adversary $\mathbf{Adv}_H^{coll}(B)$ should be very small.

Shamir proposes a variant of the discrete log hash that is believed to be collision resistant [1]. Let g be an element of maximum order in Z_k^* (i.e., an element of order $\lambda(k) = \mathsf{lcm}(r-1, s-1)$. Assume that k and g are fixed and public, r and s are secret large primes.

$$f(x) = g^x \mod k$$

[1] http://diswww.mit.edu/bloom-picayune/crypto/13190

As shown in [71], the above function has the collision resistant properties.

We can also heuristically develop the hash function by showing that the function is secure to some security attacks. For example, the hash function SHA-256 is heuristically built and shown no attack, for example preimage attack, and collision attack [72–74].

A widely used method to develop the hash function is by combining a compression function that works for a fix length block (i.e., $C : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$) and a domain extender that support any length of message. The compression function C can be constructed using a similar method used to construct the block ciphers in the symmetric encryption algorithm. The popular domain extender, called Merkle-Damgard construction, works for a message M using the following steps:

Algorithm 2: Merkle-Damgard based hash function

```
Break $M$ into $m$-bit blocks $M_1, ..., M_k$
Set $h_0 = IV$, where $IV$ is an $n - bit$ initialization vector
for $i = 1$ to $k + 1$ do
    $h_i \leftarrow C(h_{i-1}, M)$
end for
return $h_{k+1}$
```

2.4.1.2 Private Key Encryptions

A private key encryption algorithm transforms a message and a secret key into an intelligible ciphertext that can only be decrypted back to the original message by using a decryption algorithm and the knowledge of the secret key. The main drawback of the private key encryption is we also need to transfer the key securely to the receiver. However, the private key encryptions use more simple operations, so that generally, the private key encyptions are more efficient than the public key encryptions. Formally, a private key encryption scheme consists of three algorithms as follows [68, 75]:

Definition 2.8. A private key encryption scheme consists of three probabilistic polynomial-time algorithms (Gen,Enc,Dec) where:

1. Gen takes a security parameter k and return a key $K \leftarrow \mathsf{Gen}(k)$.

2. Enc takes as inputs a key K and a plaintext message $M \in \{0,1\}^*$, and return a ciphertext $C \leftarrow \mathsf{Enc}_K(M)$.
3. Dec takes a inputs a key K and a ciphertext C, and outputs the plaintext message $M \leftarrow \mathsf{Dec}_K(M)$, so that $M \leftarrow \mathsf{Dec}_K(\mathsf{Enc}_K(M))$ for all $M \in \{0,1\}^*$.

The classical cryptosystem uses some simple transformations (substitutions and permutations) to transform the message into unintelligent form. For example Caesar and Vigenere ciphers convert the message by shifting the alphabet by using a key [76, 77]. Modern cryptosystem uses more complex transformations, for example the block ciphers DES and AES [78, 79] uses S-Boxes and larger permutation to transform the messages.

The Data Encryption Standard (DES) uses Feistel method to transform the messages, so that we can have a similar algorithm for encryption and decryption. Basically, the Feistel ciphers transform the messages by executing many rounds, where each round $Round_i$ takes $2m$ bits defined as follows [80]:

$$Round_i : (L_i, R_i) \rightarrow (R_{i-1}, F(K_i, R_{i-1}) \oplus L_{i-1})$$

where L_i and R_i are the left and right part with length m each and K_i is the round key that are generated by a key schedule function. In DES, F is a function that produces m bits value using the following stages: expansion and key mixing, substitutions with S-boxes and permutations.

The Advanced Encryption Standard (AES) is also an iterated block cipher that consists of repeated application of the round transformation. AES does not use Feistel method, it uses the Substitution-Permutations Network (SPN) to transform the messages. The SPN transformations should be invertible to implement the decryption algorithm, while in the DES, the F function (because of Feistel method) does not need to be invertible. Each round of AES consists of four main operation [81]: (1) AddRoundKey (2) MixColumn (3) SubBytes (4) ShiftRows.

To encrypt a large data using a block cipher, we need to break the data into many blocks and use a mode of operation to encrypt the data. The simple Electronic Codebook (ECB) mode is not secure because the result is deterministic (the same data will produce the same ciphertext). The provable secure mode of operations, for example the Cipher Block Chaining (CBC) and Counter (CTR), produce the ciphertexts that are difficult to distinguish to the random sequence. CBC and CTR modes encrypt the large messages by using the following methods:

1. In CBC, we need to have a random initialization vector (IV). Each encrypted block (c_i) using CBC is defined as $c_i = E(k, p_i \oplus c_{i-1}), c_0 = IV$ [82]. To decrypt a block c_i we use the formula: $p_i = E(k, c_i) \oplus c_{i-1}, c_0 = IV$.

2. The CTR uses a nonce n (a one time used random number) and counter i to encrypt each block of the message. Each block encrypted with CTR is defined as $c_i = E(k, n \oplus i) \oplus p_i$ [82]. To decrypt a block c_i we use the formula: $p_i = E(k, n \oplus i) \oplus c_i$.

2.4.1.3 Public Key Encryptions

A breakthrough in cryptography is the invention of public key encryption. The public key encryption solves an inherent problem in symmetric key encryption, that is the sender should share a private key to the receiver. In a public key encryption scheme, the encryption key is public, so that the keys can be safely accessed by any parties. Most public key cryptosystems are not as fast as the private key cryptosystem, because the operations use more costly mathematical computations. In practice, most systems uses the hybrid methods, where the encryption keys are shared using a public key encryption scheme, while the data is encrypted using a private key encryption scheme.

A public key encryption scheme also uses three algorithms (Gen, Enc, Dec). The main characteristic of the public key encryption scheme is that the algorithms use different keys for encryption and decryption [68].

Definition 2.9. A public key encryption scheme consists of three probabilistic polynomial-time algorithms (Gen,Enc,Dec) where:

1. Gen takes a security parameter k and return a pair of keys $(pk, sk) \leftarrow \mathsf{Gen}(k)$.

2. Enc takes as inputs a public key pk and a plaintext message $M \in \{0,1\}^*$, and return a ciphertext $C \leftarrow \mathsf{Enc}_{pk}(M)$.

3. Dec takes as inputs a secret key s and a ciphertext C, and outputs the plaintext message $M \leftarrow \mathsf{Dec}_{sk}(M)$, so that $M \leftarrow \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M))$ for all $M \in \{0,1\}^*$.

An example of the public key cryptosystem is the RSA algorithm proposed by Rivest, Shamir and Adleman [83]. To generate the public and private keys we choose two very large and random primes number p and q and compute $n = pq$. Then choose d, that is a large and random integer which is relatively prime to $(p-1)(q-1)$. The integer e is computed from p, q, and d by finding multiplicative inverse of d modulo $(p-1)(q-1)$, so that $ed \equiv 1 \mod (p-1)(q-1)$. The multiplicative inverse can be computed by using the Extended Euclidean algorithm. The pair of positive integers (e, n) is the public key, while (d, n) is the private key. To encrypt m where $0 \leq m \leq (n-1)$, we compute ciphertext $c \equiv m^e \mod n$. To convert the ciphertext c back to the message m, compute $m \equiv c^d \mod n$.

The scheme is correct because for the Euler totient function $\varphi(n)$ (where $n = pq$) for any integer message m which is relatively prime to n, then $m^{\varphi(n)} \equiv 1 \mod n$. Because $\varphi(n) = \varphi(p)\varphi(q)$, $\varphi(p) = (p-1)$, and $\varphi(q) = (q-1)$ for prime numbers p and q, then $\varphi(n) = (p-1)(q-1)$. In the RSA algorithm, e is a multiplicative inverse of d modulo $(p-1)(q-1)$, so that $ed \equiv 1 \mod \varphi(n)$, and we should be able to find a non negative integer h, where $ed = 1 + (h\varphi(n))$. Because $c \equiv m^e \mod n$, then $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{1+(h\varphi(n))} \equiv m \mod n$.

Another example of the public key encryption scheme is a discrete log based encryption (Elgamal encryption) [84]. In a group G of order q that has a primitive root g, compute the private key as a random value x where $1 < x < q$, and the public key is $y = g^x \mod p$. To encrypt the message m, generate a random k where $1 < k < q$, then compute $c_1 = g^k \mod q$, and calculate $s \equiv y^k \equiv g^{xk}$. Then, calculate $c_2 \equiv m \cdot s$, to produce the ciphertext (c_1, c_2). To decrypt the ciphertext, first the receiver computer the shared secret $s \equiv c_1^x$, the computes $m \equiv c_2 \cdot s^{-1}$ where s^{-1} is the inverse of s in the group G.

The other public key cryptosystems are Paillier, Cramer-Shoup and Boneh-Franklin schemes. The Paillier scheme [85] uses another form of assumption about the complexity of computation in the number theory to prove the security of the scheme. The assumption states that given a composite n and an integer z it is difficult to decide whether there exists y such that $z \equiv y^n \mod n^2$. Cramer-Shoup encryption scheme [86] is an extension to the ElGamal algorithm by improving the original scheme so that it is secure under Chosen Ciphertext Attack (CCA). Boneh-Franklin scheme is a type of public key encryption scheme that supports the usage of the unique identity (id) of a user as the public key [87].

2.4.1.4 Digital Signatures

A digital signature scheme is implemented to verify the integrity and authenticity of a document. The digital signature can be implemented by using a variant of a public key encryption scheme, for example in the RSA signature, a party can sign a message by encrypting the message using the private key. The digital signature scheme include a verification algorithm that takes the the public key, the signature and the signed message to check the integrity and authenticity of the messages.

Definition of a digital signature scheme [68, 88, 89]:

Definition 2.10. A digital signature scheme consists of three probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ where:

1. Gen takes a security parameter k and return a pair of keys $(pk, sk) \leftarrow \mathsf{Gen}(k)$.
2. Sign takes as inputs the secret key sk and the message $M \in \{0,1\}^*$, and return a signature $\sigma \leftarrow \mathsf{Sign}_{sk}(M)$.
3. Verify takes as inputs the message M and a signature σ, and outputs a bit $(\mathsf{valid}, \mathsf{invalid}) \leftarrow \mathsf{Verify}_{pk}(M, \sigma)$, so that $\mathsf{Verify}_{pk}(M, \mathsf{Sign}_{sk}(M)) = \mathsf{valid}$ for all $M \in \{0,1\}^*$.

In the RSA signature scheme, for the RSA public and private key pair (e, n) and (d, n), we compute hash $h = H(m)$, and produce the signature $\sigma \equiv h^d \mod n$. To verify the signature σ, we compute $h = H(m)$ and check whether $\sigma^e \equiv h \mod n$.

Another popular signature scheme is Elgamal signature [90]. In a group $\mathbb{Z}_p$ that has a primitive root g, compute the private key as a random value x where $1 < x < p$, and the public key is $y = g^x \mod p$. To sign the message, generate a random k, then compute $r = g^k \mod p$ where $r \neq 0$, and calculate $s \equiv k^{-1}(H(m) - xr) \mod p - 1$, for a secure hash function H. The signature is (r, s). Signature verification works by checking whether $g^{H(m)} \equiv y^r r^s \mod p$.

Schnorr proposed a signature scheme as follows [91]: let q be a large prime, and p a larger prime such that $p \equiv 1 \mod q$. Let g be a generator of a cyclic group of order q in $\mathbb{Z}_p$. For a random secret key x, the public key is $g^x \mod p$. The signature for a message m is produced by first compute $r = g^k \mod p$ for a random k, produce $h = H(m, r)$, compute the signature $s = k + hx \mod q$, and return (h, s) as the signature. Digital Signature Algorithm (DSA) can be viewed as combination of the Elgamal and the Schnorr signature [92].

Boneh et al. proposed a short signature by assuming the existence of bilinear maps $e : G_1 \times G_2 \rightarrow G_T$ where for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}_{|}$ and $e(u^a, v^b) = e(u, v)^{ab}$[93]. For a generator g_1 in G_1, a generator g_2 in G_2, and a random secret key $x \in \mathbb{Z}_{|}$, compute the public key $v = g_2^x \in G_2$. The signature $\sigma_i \in G_1$ on the message m_i is produced by computing $\sigma_i = H(m_i)^x$. By having bilinear maps $e : G_1 \times G_2 \rightarrow G_T$, we can verify whether $e(\sigma_i, g_2) = e(h, g_2^x)$.

Many signatures can be aggregated into one signature for more efficient space and verification. Aggregate signature is a technique to combine signatures on many different messages into a short signature. Some aggregate signatures have restriction that they can only be verified if there is no duplicate messages or public keys. However, it is possible to develop a scheme that does not have any restriction [94].

For the RSA signature, the basic method of aggregation is by computing the product of the signatures as follows. For signatures $\sigma_0, \sigma_1, ...\sigma_{t-1}$ the signature can be condensed into a signature σ by computing [95]:

$$\sigma = \prod_{i=0}^{t-1} \sigma_i \mod n$$

The method can also be used for the signature based on pairing [96], that is with the requirement of the existence of a mapping between groups for example the map $e : G_1 \times G_2 \rightarrow G_T$ where $|G_1| = |G_2| = |G_T|$ with bilinear (for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{ab}$) and non-degenerate ($e(g_1, g_2) \neq 1$) properties. A particular aggregate signature scheme proposed by Boneh et al. [96] is as follows:

Key Generation the user picks random secret key $x \overset{R}{\leftarrow} \mathbb{Z}_p$ and computes the public key $v \leftarrow g_2^x$.

Signing to sign a message m_i, compute $h_i \leftarrow H(m)$, where $h \in G_1$, and the signature $\sigma_i \leftarrow h_i^x$.

Aggregation for a set of signatures $\{\sigma_1, \sigma_2, ..., \sigma_k\}$, compute the aggregate signature $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$.

Aggregate Verification for all users $u_i \in U$ with public keys $v_i \in G_1$ and the original messages m_i, computes $h_i \leftarrow H(m_i)$ and accept if $e(\sigma, g_2) = \prod_{i=1}^{k} e(h_i, v_i)$ holds.

We may see that $e(h_i, v_i) = e(h_i, g_2^x)$, and $e(\sigma_i, g_2) = e(h_i^x, g_2)$. Because of the pairing property, $e(h_i, g_2^x) = e(h_i^x, g_2) = e(h_i, g_2)^x$.

Based on the work of Boneh et al. [96], Bellare et al. analyze the workaround suggested Boneh et al. regarding the restriction that all messages $m_1, ..., m_n$ should be distinct by appending the public key g^x to each message m, so in the signing step we compute signature $\sigma \leftarrow h^x$, where $h \leftarrow H(g^x || m)$. Bellare et al. showed the requirement that $g^{x_1} || m_1, ..., g^{x_n} || m_n$ should be distinct can be removed without compromising the security [94].

2.4.2 How to Prove the Security of Cryptosystems

2.4.2.1 Security Reduction

A fundamental question in the computer security is how to prove the property of a security scheme. For example, an encryption scheme should be proved to be secure under all possible security attacks. To prove a security scheme, we often need to prove the "inexistence", rather than the "existence" of the conditions. The method to prove the security of the cryptosystem is often difficult to grasp for normal computer science researchers because it has different characteristics to the method to prove the correctness of algorithms. The method of "proof by contradiction" is often used where the researchers prove that if the attacker successfully attacks the system, the attacker can also solve some known hard problems.

To prove the security property of a cryptosystem, first we need to formally define the security properties that capture the requirements where the scheme will be used in the real life. For example, an attack to an encryption system is extracting information about the plaintext or key from the ciphertext. Another attack is distinguishing two ciphertext which can be used to extract more information about the plaintext or the key. Attack to signature scheme is forgery, for example finding a valid pair of message and signature without having the private key. The security proof needs to show that these attack is not possible to be executed under the assumptions about the power of the adversary.

To prove the security property of a cryptographic scheme, we can use reduction method, where we need to find a reduction of the problem to attack the cryptosystem to the problem of solving some hard problems (for example the RSA or discrete log problem). Bellare described the reduction as follows [97].

> Here is another way of looking at what reductions do. When I give you a reduction from the one-wayness of RSA to the security of my protocol, I am giving you a transformation with the following property. Suppose you claim to be able to break my protocol. Let P be the program that does this. My transformation takes P and puts a simple "wrapper" around it, resulting in a protocol P'. This protocol P' provably breaks RSA. Conclusion? As long as we believe you can't break RSA, there could be no such program P . In other words, my protocol is secure.
>
> Those familiar with the theory of NP-completeness will recognize that the basic idea of reductions is the same. When we provide a reduction from SAT to some problem we are saying our problem is hard unless SAT is easy; when we provide a reduction from RSA to our protocol, we are saying the latter is secure unless RSA is easy.

2.4.2.2 Proofs in the Random Oracle Model

Random oracle model is a model to analyze the security schemes by assuming the existence an idealized random oracle [98]. The random oracle should be able to return perfectly random outputs (the assumption which is somewhat cannot be truly applied in the real world). The random oracle model was introduced as a more practical way to prove the security [98].

It is often easier to prove the security of the crypto schemes by using idealized security primitives (random oracle for the hash function, or ideal cipher for encryption), because we do not need to analyze the inner working of the security primitives. However, the primitives that are used in the real cryptosystems are not ideal, so that it is desirable to have a security proof without the assumption about existence of the random oracle. The proof without the requirement of the existence of the random oracle and other idealized functions is called the standard model. In the standard model, the security proofs may still use the assumptions about the computation complexity of some difficult problems.

2.4.2.3 Attack Scenarios

The attack scenarios to an encryption scheme can be categorized into following:

- *Ciphertext-only attack (COA).* In this scenario, the attacker can only observe the ciphertext. No access to the plaintext. The purpose of attack is to determine the plaintext.
- *Known-plaintext attack (KPA).* The attacker knows some pairs of plaintext and ciphertext encrypted under the same key. The attacker attempts to determine the plaintext that is encrypted in other ciphertext.
- *Chosen-plaintext attack (CPA).* The attacker has capability to obtain the ciphertext for his/her chosen plaintext. The attacker tries to find the plaintext encrypted in other ciphertext.
- *Chosen-ciphertext attack (CCA).* In this scenario, the attacker can obtain decryption of his/her chosen ciphertext. The aim of attack is to determine the plaintext correlated to other ciphertext.

A modern encryption scheme should be shown to be secure at least in the CPA model, and it is desirable to be proved secure in the CCA model.

In a signature scheme, the attack scenarios are as follows:

1. *Total break*: disclosing the signer's private key
2. *Universal Forgery*: constructing an algorithm which can sign any messages
3. *Existential Unforgeability*: Providing a new message-signature pair.

A modern digital signature requires the security under the strongest attack, that is existential unforgeability (EUF). In the EUF definition, a digital signature is secure if no pair of message and the signature can be created without access to the private key [88]. In the model, the attacker is provided access to the pair of messages and the signatures for any chosen messages, and by using these information the attacker adaptively tries to forge the signature by creating a new valid message the signature. This security model is often described as EUF-CMA (Existential Unforgeability under Chosen Message Attack). The EUF-CMA can be written in the following algorithm (Algorithm 3):

Algorithm 3: Existential Unforgeability under Chosen Message Attack

```
$(pk, sk) \leftarrow Gen(1^k)$/* Global Vars */
$r \overset{r}{\leftarrow} R$, view $\leftarrow \{r, pk\}$
$OracleQueries(A, view)$
$(m^*, \sigma^*) \leftarrow A(view)$
**return** $ver_{pk}(m^*, \sigma^*)$/* return 1 if the signature is valid, 0 otherwise */
**function** $OracleQueries(A, view)$
  **loop**
    $m \leftarrow \{A, view\}$,
    $\sigma \leftarrow SigningOracle(m)$,
    $view \leftarrow view \cup \{\sigma\}$
  **end loop**
**end function**
**function** $SigningOracle(m)$
  $\sigma \leftarrow sig_{sk}(m)$
**return** $\sigma$
**end function**
```

As shown in Algorithm 3, the attacker has access to an oracle that can generate a valid pair of message and signature upon request by the attacker (the chosen messages). Then the attacker tries to forge at least a valid pair of message and signature. The attacker successfully attacks the signature if he/she can show a pair of message and signature which has not been previously requested to the oracle.

2.4.2.4 Game-based Security Proof

The attacks in the cryptosystem can be represented as a game between the adversary and the challenger, where the adversary tries to attack the scheme (under the scenarios described in Section 2.4.2.3), and the challenger interacts with the adversary and provides the challenges that need to solved by the attacker. If the adversary solves the challenges, then the adversary wins the game and the security of the scheme is compromised. Both of the adversary and challenger are probabilistic processes that communicate with each other.

We can construct the security proof by identifying the event S that represents the possibility of the success of the adversary (for example, the event that adversary forges the signature, or distinguishing two encrypted text). A cryptosystem is defined secure if for each adversary the probability of the event S is close to a specific value (i.e., 0 or $\frac{1}{2}$). For example, the ciphertext indistinguishability under chosen plaintext attack (also called the *semantic security*) game for a public key encryption scheme is defined as the game between adversary and challenger so that the adversary cannot infer any information about the plaintext from its ciphertext for the next encryption, even if the adversary knows previously encrypted form of some chosen messages. The semantic security game is described in Algorithm 4.

Algorithm 4: Semantic Security

```
$(pk, sk) \leftarrow Gen(1^k)$/* Global Vars */
$r \xleftarrow{r} R$, $view \leftarrow \{r, pk\}$
$(m_0, m_1) \leftarrow A(view)$
$b \xleftarrow{r} \{0,1\}, c \leftarrow f_{pk}(m_b), view \leftarrow view \cup \{c\}$
$\hat{b} \leftarrow A(view)$
if($\hat{b} = b$) then return 1 else return 0
```

The security proofs using game-based security proof can be described by showing a sequence of games that can be reduced to each other. Concretely, we need to construct games $G_0, G_1, ..., G_n$ where the G_0 is the original attack. For each game G_i, we need to compute the probability of event S_i and prove that $\Pr[S_i]$ is close to $\Pr[S_{i+1}]$ for $0 \leq i < n$, and the probability of event S_n is close to the probability of solving the target hard problem. By relating $\Pr[S_0]$ to $\Pr[S_1]$, and $\Pr[S_i]$ to $\Pr[S_{i+1}]$, we can prove the relation of $\Pr[S_0]$ to $\Pr[S_n]$ which complete the proof.

There are three types of reductions (also called the transitions) that are normally used in the security proof using game-based security proof method [99]:

1. Transition based on indistinguishability.
   We need to prove that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is very small by showing a distinguishing algorithm that can run for both input distribution in game G_i and G_{i+1} with probability $\Pr[S_i]$ and $\Pr[S_{i+1}]$. The different probability of occurrence of the event S_i and S_{i+1} in both games should be small.

   For example, if in the game G_i we need to distinguish three inputs (g^x, g^y, g^{xy}) mod p, the probability of success is close to the probability in game G_{i+1} to distinguish $(g^x, g^y, g^z) \mod p$ for some random z because of the Decisional

Diffie-Hellman assumption where the advantage of a distinguishing algorithm D, that is:

$$\begin{aligned} DDHAdv(D) = |&\Pr_{x,y}[D(g^x, g^y, g^{xy}) = 1] \\ &- \Pr_{x,y,z}[D(g^x, g^y, g^z) = 1]| \end{aligned}$$

is very small.

2. Transition based on failure events.

We need to show that Game i and $i + 1$ are identical unless some failure events F occurs, so that

$$S_i \wedge \neg F \iff S_{i+1} \wedge \neg F$$

The following lemma is often used.

Lemma 2.11. ***(Difference Lemma [99, 100])****. Let A,B,F be three probabilistic events such that $A \wedge \neg F \iff B \wedge \neg F$, then $|Pr[A] - Pr[B]| \leq Pr[F]$.*

Proof. Because of $A \wedge \neg F \iff B \wedge \neg F$, we get:

$$\begin{aligned} |\Pr[A] - \Pr[B]| &= |\Pr[A \wedge F] + \Pr[A \wedge \neg F] \\ &- \Pr[B \wedge F] - \Pr[B \wedge \neg F]| \\ &= |\Pr[A \wedge F] - \Pr[B \wedge F]| \\ &\leq \Pr[F] \end{aligned}$$

□

So, by showing $\Pr[F]$ is negligible we can prove that $\Pr[S_i]$ is close to $\Pr[S_{i+1}]$.

3. Bridging steps.
This type of transition is used to make the proof clearer and easy to follow by formulating the game in a different way such that $\Pr[S_i]$ =$\Pr[S_{i+1}]$.

An example of the security proof using the game-playing technique is the security proof of Cipher Block Chaining (CBC) with random initialization vector (IV). As described in Section 2.4.1.2, each encrypted block is defined as $c_i = E(k, p_i \oplus c_{i-1}), c_0 = IV$. At first, we need to define the original indistinguishability under chosen plaintext attack (IND-CPA) game in the Game **G0**.

G0. This game represents the original game. In each query q_i, the adversary A chooses two n blocks plaintexts $(M_i[0], M_i[1])$ and given access to the encryption oracle. The oracle encrypts the plaintexts and return the ciphertext, the adversary should guess whether the ciphertexts belong to the left (0) or right (1).

```
$K \xleftarrow{\$} \{0,1\}^k, b \xleftarrow{\$} \{0,1\}, S \leftarrow \emptyset$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(r, C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    $c_i[0] \xleftarrow{\$} \{0,1\}^n$
    for $j = 1$ to $n$ do
        $P \leftarrow c[i-1] \oplus m[j]$
        if $P \notin S$ then
            $\mathsf{T}[P] \leftarrow E_K(P)$
        end if
        $c[i] \leftarrow \mathsf{T}[P]$
    end for
    $S \leftarrow S \cup \{P\}$
end for
$d \leftarrow A(r, C_1, ..., C_q)$
return $(b = d)$
```

Let $\Pr[Guess \Rightarrow true]$ be the probability that the adversary correctly guess the left or right oracle, then:

$$\mathbf{Adv}_{\mathcal{SE}}^{lor-cpa-b}(A) = 2 \cdot \Pr[Guess \Rightarrow true] - 1$$

Proof:

$$\begin{aligned}
&\Pr[Guess \Rightarrow true] \\
&= \Pr[Guess \Rightarrow 1 | b = 1] \cdot \frac{1}{2} + \Pr[Guess \Rightarrow 1 | b = 0] \cdot \frac{1}{2} \\
&= \Pr[Guess \Rightarrow 1 | b = 1] \cdot \frac{1}{2} + \left(1 - \Pr[Guess \Rightarrow 0 | b = 0] \cdot \frac{1}{2}\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[Guess \Rightarrow 1 | b = 1] - \Pr[Guess \Rightarrow 0 | b = 0]) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\mathbf{Adv}_{\mathcal{SE}}^{lor-cpa-b}(A)\right)
\end{aligned}$$

Because $\Pr[Guess \Rightarrow true]$ is the condition where the adversary wins the game **G0**. So, that

$$\mathbf{Adv}^{lor-cpa-b}_{\mathcal{PR}}(A) = 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1$$

G1. In this game, we assume E is pseudorandom function with a PRF advantage $\mathrm{Adv}_E^{prf}(B)$, and we now use the random values rather than E, so that the difference of the advantages of **G0** and **G1** is:

$$\Pr[G_0^A \Rightarrow 1] - \Pr[G_1^A \Rightarrow 1] = \mathsf{Adv}_E^{prf}(B)$$
$$\Pr[G_0^A \Rightarrow 1] = \Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_E^{prf}(B)$$

```
$K \xleftarrow{\$} \{0,1\}^k, b \xleftarrow{\$} \{0,1\}, S \leftarrow \emptyset$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(r, C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    $c_i[0] \xleftarrow{\$} \{0,1\}^n$
    for $j = 1$ to $n$ do
        $P \leftarrow c[i-1] \oplus m[j]$
        if $P \notin S$ then
            $\mathsf{T}[P] \xleftarrow{\$} \{0,1\}^n$
        end if
        $c[i] \leftarrow \mathsf{T}[P]$
    end for
    $S \leftarrow S \cup \{P\}$
end for
$d \leftarrow A(r, C_1, ..., C_q)$
return $(b = d)$
```

G2. In this game, we modify the game **G1** and define the event $\mathsf{bad} \leftarrow \mathsf{true}$ whenever there is collision on $P \leftarrow c[i-1] \oplus m[j]$. In game **G2**, whenever there is collision on P, we will use the consistent value (previously computed) as P.

$$\Pr[G_1^A \Rightarrow 1] = \Pr[G_2^A \Rightarrow 1]$$

```
$K \xleftarrow{\$} \{0,1\}^k, b \xleftarrow{\$} \{0,1\}, S \leftarrow \emptyset$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(r, C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    $c_i[0] \xleftarrow{\$} \{0,1\}^n$
    for $j = 1$ to $n$ do
        $P \leftarrow c[i-1] \oplus m[j]$
        $c[i] \xleftarrow{\$} \{0,1\}^n$
        if $P \in S$ then
            bad $\leftarrow$ true
            $c[i] \leftarrow \mathsf{T}[P]$
        end if
        $\mathsf{T}[P] \leftarrow c[i]$
    end for
    $S \leftarrow S \cup \{P\}$
end for
$d \leftarrow A(r, C_1, ..., C_q)$
return $(b = d)$
```

G3. The game **G3** is exactly the same as game **G2**, except that in game **G3**, whenever there is collision on P, the game stops and the adversary wins.

```
$K \xleftarrow{\$} \{0,1\}^k, b \xleftarrow{\$} \{0,1\}, S \leftarrow \emptyset$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(r, C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    $c_i[0] \xleftarrow{\$} \{0,1\}^n$
    for $j = 1$ to $n$ do
        $P \leftarrow c[i-1] \oplus m[j]$
        $c[i] \xleftarrow{\$} \{0,1\}^n$
        if $P \in S$ then
            bad $\leftarrow$ true
        end if
        $\mathsf{T}[P] \leftarrow c[i]$
    end for
    $S \leftarrow S \cup \{P\}$
end for
$d \leftarrow A(r, C_1, ..., C_q)$
return $(b = d)$
```

So that, $\Pr[G_2^A \Rightarrow 1]$ is the same as $\Pr[G_3^A \Rightarrow 1]$ until $\mathsf{bad} \leftarrow \mathsf{true}$.

$$\Pr[G_2^A \Rightarrow 1] \leq \Pr[G_3^A \Rightarrow 1] + \Pr[G_3^A \text{ sets } \mathsf{bad}]$$

And, because **G3** only uses the random values as the values of $c[i]$, then the probability of the adversary wins the game is

$$\Pr[G_3^A \Rightarrow 1] = \frac{1}{2}$$

G4. In game **G4**, we directly set the value of P as a random value. We argue that game **G4** is the same as **G4** because in **G3**, $P \leftarrow c[i-1] \oplus m[j]$ while $c[i] \xleftarrow{\$} \{0,1\}^n$.

```
$K \xleftarrow{\$} \{0,1\}^k, b \xleftarrow{\$} \{0,1\}, S \leftarrow \emptyset$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(r, C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    $c_i[0] \xleftarrow{\$} \{0,1\}^n$
    for $j = 1$ to $n$ do
        $P \xleftarrow{\$} \{0,1\}^n$
        $c[i] \xleftarrow{\$} \{0,1\}^n$
        if $P \in S$ then
            bad ← true
        end if
    end for
    $S \leftarrow S \cup \{P\}$
end for
$d \leftarrow A(r, C_1, ..., C_q)$
return $(b = d)$
```

So, that:

$$\Pr[G_3^A \text{ sets } \mathsf{bad}] = \Pr[G_4^A \text{ sets } \mathsf{bad}]$$

The probability of the collision on a set of n-bits random values for q queries should be less than $\frac{q^2}{2^{n+1}}$, so that

$$\Pr[G_4^A \text{ sets } \mathsf{bad}] \leq \frac{q^2}{2^{n+1}}$$

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{PR}}^{lor-cpa}(A) &= 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1 \\
&\leq 2 \cdot \Big(\Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)\Big) - 1 \\
&\leq 2 \cdot \Big(\Pr[G_2^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)\Big) - 1 \\
&\leq 2 \cdot \Big(\Pr[G_3^A \Rightarrow 1] + \Pr[G_3^A \text{ sets } \mathsf{bad}] + \mathsf{Adv}_F^{prf}(B)\Big) - 1 \\
&\leq 2 \cdot \left(\frac{1}{2} + \Pr[G_4^A \text{ sets } \mathsf{bad}] + \mathsf{Adv}_F^{prf}(B)\right) - 1 \\
&\leq 2 \cdot \left(\frac{1}{2} + \frac{q^2}{2^{n+1}} + \mathsf{Adv}_F^{prf}(B)\right) - 1 \\
&\leq 2 \cdot \mathsf{Adv}_F^{prf}(B) + \frac{q^2}{2^n}
\end{aligned}$$

2.4.2.5 Simulation-based Security Proof

In the simulation-based security proof, we need to show that the attacker who interacts with a simulator in the ideal world is indistinguishable from the challenger in the real world [101]. The simulator is assumed to be in ideal world where the scheme is secure by definition, so that by definition the simulator cannot attack the system, and has no access to the private information. If we can show that the simulator can interact with the attacker without being detected by the attacker, and exploit the attacker (i.e., treats the attacker as a sub-routine) to solve the difficult problem, we can prove the security of the scheme.

For example, to prove the security of a signature scheme, we need to prove that if the attacker A can forge the signature in *EUF-CMA*, then we show that there exists a simulator B that can simulate A's environment without detected by A and then by using A's outputs, the simulator B can solve the RSA problem. Because the simulator does not have access to the private information in the RSA signature and we believe that RSA problem is hard, this is a contradiction.

In the following, we provide an example of the proof of the Full Domain Hash (FDH) signature [102]. In the FDH, for the RSA public and private key pair (e, n) and (d, n), we compute hash $h = H(m)$, and produce the signature $\sigma \equiv h^d \mod n$. To verify the signature σ, we compute $h = H(m)$ and check whether $\sigma^e \equiv h \mod n$. The basic assumption is the hash function H maps the inputs uniformly into the full domain of Z_N in the RSA function.

In the EUF-CMA model, the attacker A is given access to the random oracle H and the signing function $Sign$, simulator B should be able to simulate both of the random oracle and the signing function without having better information than A. A should produce a valid forgery m_i* and j_i* and σ_i* and we need to show how B uses A's outputs to solve an RSA problem. To prove security of FDH, we shows how B simulates these functions without having access to the RSA private keys.

Setup. The simulator B is given the signer public key (e, N), but B has no access to the private key d. The simulator B is also provided with y and later B should solve the RSA problem by computing $y^d \mod N$ from A's outputs. The simulator B starts by giving the forger algorithm A the public key (e, N).

Answering Signature Queries.

To answer the signature queries for the message m_i, B sets a random signature x_i and sets $H(m_i) = x_i^e \mod N$. The simulator should keep the pair of messages and the hashes, so that the values can be retrieved later.

Answering H Queries. To answer the hash queries for the message m_i, B picks a random x_i

1. with a probability p, the simulator returns $H(m_i) = x_i^e \mod N$
2. with a probability $1 - p$, the simulator returns $H(m_i) = y \cdot x_i^e \mod N$

After answering the queries, the simulator should also keep the pair of messages and the hashes.

Output.

To forge the signature, the forger outputs a signature message pair $(\sigma_i^\star, m_i^\star)$ that has not been queried before. With probability $1-p$, the $H(m_i^\star) = y \cdot (x_i^\star)^e \mod N$ and $\sigma_i^\star \equiv (y \cdot (x_i^\star)^e)^d \equiv y^d \cdot x_i^\star \mod N$. The simulator can easily output $y^d \equiv \sigma_i^\star / x_i^\star$.

From the above argument, we show that if there exists a forger A who breaks our signature scheme then there exists a simulator B that can solve the RSA problem.

Chapter 3

An Integrity Scheme for the Provenance Recording System

3.1 Introduction

Provenance of an object is the documentation about the origin and how to produce the object [1, 18, 32–35]. It describes the causal relationship between objects, processes and the actors that control the processes and objects. For example, in a hospital, the provenance describes the causal relationships between the objects (i.e., medical records, medical test results), the processes that produce the objects (i.e., a medical checkup, medical diagnosis) and the actors that control the processes (i.e., the physicians). In grid and cloud systems, the provenance records the source of the objects and the processes that affects the condition of objects produced in the system. The provenance is important to verify the quality of the processes and objects.

The provenance can be explicitly recorded and stored along with the objects in the same or different file systems/databases. It can also be later inferred, for example, by asking the actors that control the processes or by checking the computer logs where the processes are executed. Recently, there is much interest in explicit provenance recording. Many implementations represent the provenance as a directed graph where the nodes represent the entities (i.e., objects, processes and actors) and edges represent the causal relationship between entities [42, 43].

In a distributed system, the provenance is normally stored in a persistent storage (for example a database or a file system), where an interested user can query the provenance to verify the quality of objects produced in the system. Ideally, the storage and the computing environment should be trusted. However, with current technology in computer and network security, it is difficult (if not impossible) to implement a fully trusted storage and computing environment.

3.1.1 Motivation

A provenance recording system receives the assertions submitted by the process executors and keeps the assertions to be accessed later by the auditors. The collection of the assertions (i.e., the provenance) confirms which process executors that should be responsible to the processes and outputs. It can also confirm that some process executors are responsible for faulty processes that produce erroneous outputs. The faulty processes and outputs may cause disadvantages to the process executors who are responsible to the processes and outputs, because the parties who are affected by the outputs of the faulty processes may send complaints to the process executors.

The process executors may got reward (be respected) for high quality outputs, but they also may get disadvantages (penalty) if they produce low quality outputs. At the time of submission, the process executors may have no idea about the faulty outputs, and only realized later after the submission. This is a normal characteristic of the human being and also processes designed by human (including the processes that are executed by machines/computers). Even after careful design and execution of a process, there are possibilities of errors that can only be detected later (we may refer to many recall of the products of large/established companies because of defects in the products).

The honest process executors will take responsibility to the faulty processes and all of the consequences (including bad reputation), and try to improve their credibility later. However, the malicious ones may try to avoid the responsibility by trying to hide the provenance of the faulty processes. The malicious process executors attack the provenance which was previously submitted by them to change the facts about the faulty processes that are recorded in the provenance by updating or deleting the provenance. In a distributed processes, the attack may affect other process executors (the honest ones), because the malicious process executors may try to change the responsibility of the faulty processes to the honest ones.

There are some integrity and confidentiality mechanisms that can be employed to protect the provenance. The basic integrity mechanism is the digital signature that proves the originality of the documentation. It detects unauthorized updates to the provenance (i.e., updates by the person who is not authorized to sign the documentation). It also prevents repudiation by the signer. However, the signature cannot detect malicious updates by an "authorized" person. That is the person who owns the private key for signing the documentation.

This attack ("authorized" update) is viable because normally the parties who are interested in the provenance do not have prior knowledge or copy of the provenance. So, they do not have any evidence about the malicious but "authorized" update that has been made to the provenance. It is not efficient and also costly for each user (i.e., the auditor) who is interested in the provenance to make a copy of the provenance promptly after the provenance is submitted to the database. It is also possible that the interested users do not have any previous access to the provenance system, so the users could not make any copies of the provenance.

We show some examples of this attack. The first example is in the process of audit by an external auditor in a company where normally the documentations of processes are kept by the company. When the external auditor inspects the company, without security mechanisms that detect the alteration, the company can re-create a fresh and verifiable provenance. The external auditor cannot detect the alteration because it is signed by authorized parties (if the provenance is created by the people outside the company, they can also collude to alter the provenance). In the context of computer systems, an example is when a user wants to verify the quality of outputs of a grid system in other organizations where the user does not have access previously. Without a secure provenance system, the user does not have a choice other than believing that the provenance is correct and is not altered by "authorized" person in the organization. In the context of the medical record, the medical data of a patient is normally stored in the health care provider of the patient (i.e., the hospital). The hospital can easily change the data without the patient's consent (although the laws in many countries mandate that the data should be under the patient control).

3.1.2 The Problems of "Inconsistent Claims" and "Inconsistent Interpretations"

Due to its liquidity, the digital form of provenance is vulnerable to security problems because it can be easily copied, changed, added or deleted by anybody who has access to (either legal or illegal) the provenance database. The alteration of the provenance record may cause the integrity problem we call "inconsistent claims" and the "inconsistent interpretations" problems.

For example in Figure 3.1, *Process Executor 2* uses the output of the *Process Executor 1*, so that in this case, the output of the *Process Executor 1* affects the output of the *Process Executor 2*. Because the *Process Executor 1* is responsible to the data used by *Process Executor 2*, the *Process Executor 1* should also be responsible in part to the data output produced by *Process Executor 2*.

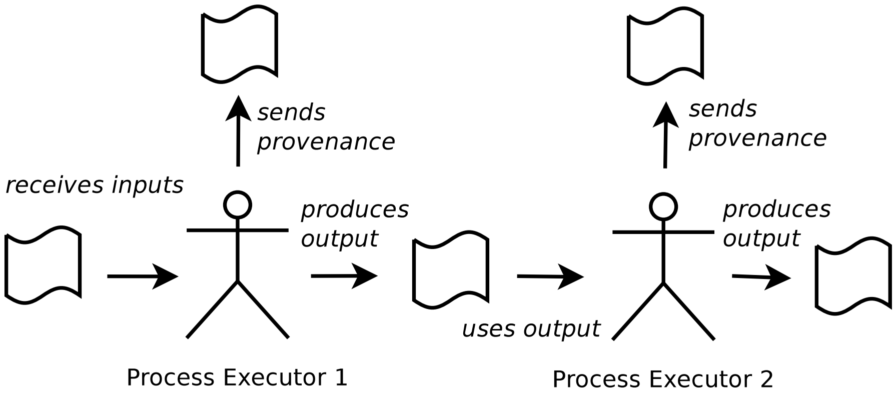

FIGURE 3.1: Provenance as a proof of responsibility of each process executor

If both process executors are honest, both of them will have consistent statements in their provenance. However, if one of them is malicious, there will be inconsistent claims between them. For example, *Process Executor 2* says that it uses an output O'_0 which was produced by *Process Executor 1*, while *Process Executor 1* may say that it never produce O'_0, its output was O'_1, not O'_0. An auditor cannot easily decide which process executor is honest, because there are two possibilities:

1. *Process Executor 1* is malicious. It wants to avoid the responsibility to its previous output O'_0 by updating the assertion (i.e., assertion about its output), and also the output O'_0 to O'_1 so that he/she is not responsible to the output of *Process Executor 2*.

2. *Process Executor 2* is malicious. He/she does not use the output O'_0 produced by the *Process Executor 1*, but claims that he/she uses an output produced by *Process Executor 1*.

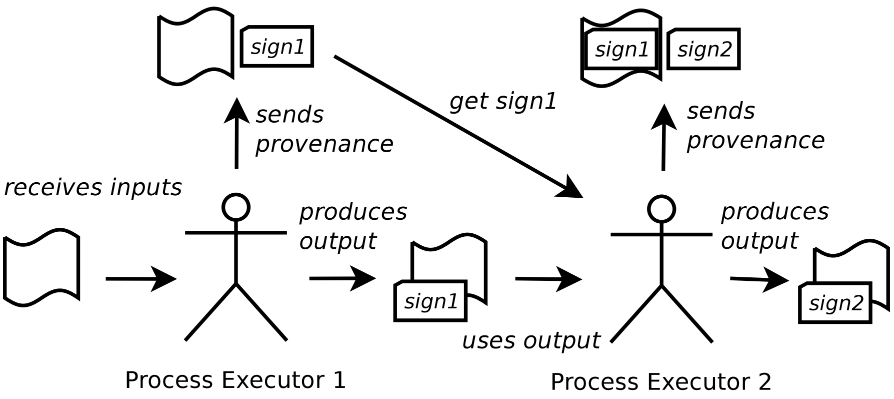

FIGURE 3.2: Signature Chain

The problem of "inconsistent interpretation" occurs when two process executors in the end of the chain collude to update the provenance, so the auditor has different interpretation to the provenance after update. For example in the above example, the *Process Executor 1* and *Process Executor 2* collude to update the provenance, so the provenance describe different assertions from the the previous/deleted versions. This attack cannot be prevented by the hash/signature chain because the integrity checking will conclude the provenance is plausible and no detected problem.

3.1.3 Contributions

In this chapter, we propose the method for the integrity mechanism for the provenance graph. Our method to protect the integrity of the provenance by employing a Trusted Counter Server (TCS) that provides a unique label which is consecutive counter for each provenance assertion in a group. By combining the counter with the signature chaining we prove that we can guarantee the security of the provenance to the consistency problems namely inconsistent claims and inconsistent interpretation attacks.

Our work extends the signature chain proposed in [5, 40, 41, 103, 104]. In our method, we employ the linking mechanism in a graph structure rather than a chain structure. We also include another layer of the integrity mechanism by employing a Trusted Counter Server (TCS). The TCS is different to a timestamping service, rather than providing the creation time for each provenance node, the TCS provides a counter number to each provenance node. In our proposed scheme, the TCS can keeps the number of nodes that can be used to detect deletion, and find which nodes that have been deleted by simply enumerating the existing nodes and find the missing counter. By using this method, we can also identify the collusion of some process executors to change some nodes that cannot be detected by the standard signature chain.

3.2 Related Work

A solution to the integrity of the provenance is by storing the provenance in a trusted storage, for example by using Write Once Read Many (WORM) storage. A CDROM device is an example of WORM implementation where the data cannot be updated after they are written to the CDROM because of physical characteristics and the method of data writing of the CDROM. However it may not convenient to use the WORM storage and also it may need a large storage to store all provenance and data which is not applicable in many situations because of the cost and performance. It is desirable to have more efficient solution in term of the trusted storage requirement.

Two methods to protect integrity of a sequence of digital documents by digital timestamping have been proposed by Habert et al. [105]. The first method employs a Trusted Time-Stamping Service (TSS) that issues signed timestamps and also links two timestamps requested consecutively. The TSS links two timestamps by storing the hash value of the first timestamp in the second timestamp. Any changes to the first timestamp can be detected by checking the hash value in the second timestamp. To produce a fake timestamp, the TSS needs to collude with all clients who are requested timestamps after the fake timestamp. The second method uses the digital signature to distribute trust among many clients. A client who needs to timestamp a document should ask k random other clients to sign the timestamp. The list of the other clients is generated by a pseudorandom generator that uses hash of the document as a seed. Because the other users are chosen randomly it is very unlikely that they collude to create a fake timestamp. This second method does not employ any TSS but assumes that the users can ask the signatures from the other users.

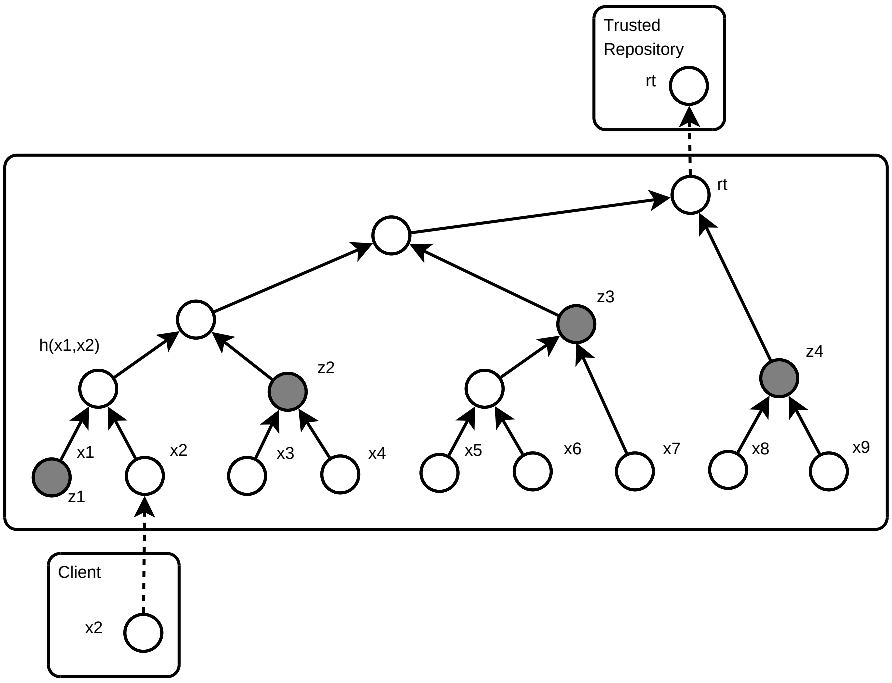

FIGURE 3.3: A simplified model of Habert and Stornetta's sceme

Buldas et al shows an improved hash-chain by using a trusted storage [106, 107]. To implement the timestamping scheme, we need to assume the existence of a write only repository R and receive data from server S in an authenticated manner. The timestamping is divided into many rounds. In a round t, server S receives timestamping requests $x_1, ..., x_m$ from its clients. After the round is over, S computes an aggregate hash $r_t = G_h(x_1, ..., x_m)$. For example, S may compute $r_t = h(x_1, h((x_2, x_3), x_4))$ and stores (t, r_t) in R.

For each request x, S issues certificate $c = (x, t, n, z)$, where t is the current time value, n is an identifier $n = n_1 n_2 ... n_l \in \{0, 1\}^l$, and z is a sequence $z = (z_1, z_2, ..., z_l) \in (\{0, 1\}^k)^l$. In Figure 3.3, the certificate for x_1 is $(x_1, t, \mathtt{0000}, (z_1, z_2, z_3, z_4))$, where $z_1 = x_2$, $z_2 = h(x_3, x_4)$, $z_3 = h(h(x_5, x_6), x_7)$, and $z_4 = h(x_8, x_9)$.

To check whether (x, t, n, z) is an authentic certificate, the verifier computes a sequence $y_0, y_1, ..., y_l$ where $y_0 = x$, and

$$y_i = \begin{cases} h(z_i, y_{i-1} & \text{if} \quad n_i = 1 \\ h(y_{i-1, z_i} & \text{if} \quad n_i = 0 \end{cases}$$

and check whether $y_l = r_t$ by querying (t, r_t) from R.

Hasan et al. [5, 40, 41] show a threat model for provenance and the method to prevent/detect the attacks associated with the threats by using digital signature, checksum and broadcast/threshold encryption. The provenance is modeled as a chain so the method cannot be applied directly to the graph model. Their method to protect integrity of the provenance chain is by signing each provenance record in the chain and including a checksum of the previous record in the current record to maintain the integrity of the records and the chain structure. They assume that no collusion of all users (the people who write provenance). Aldeco-Pérez et al. and Syalim et al. proposed similar method to secure provenance [103, 104] and applying the hash/signature-chain to the graph model.

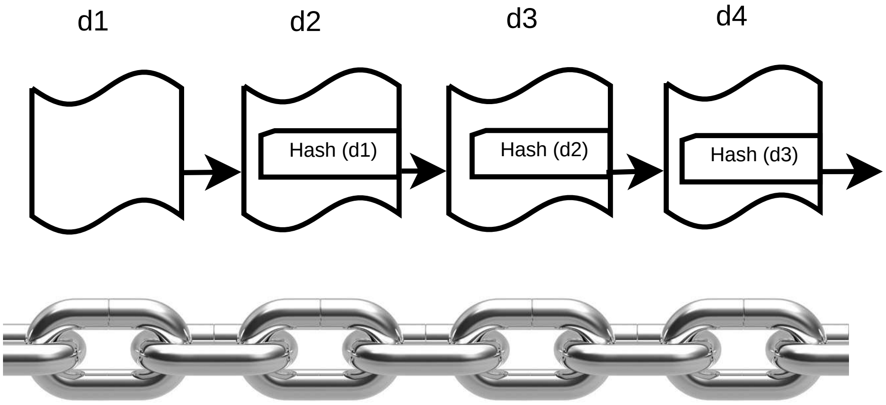

Figure 3.4: Hash Chain

The hash/signature chain as proposed in [5, 40, 41, 103, 104] works as follows (see Figure 3.4). Each process executor should sign provenance and data created by him/her. Whenever a *Process Executor 2* uses the output of *Process Executor 1*, *Process Executor 2* should keep a proof by also recording the hash/signature of the provenance and data created by *Process Executor 1* so that *Process Executor 1* cannot reject claims made by *Process Executor 2* (see Figure 3.2) if the signatures are correct. However, the hash/signature can only proof the claim if we keep the output or provenance created by *Process Executor 1* in a trusted storage. In an untrusted environment, *Process Executor 1* may easily reject the claim by updating the output and data.

The hash/signature chain can only show whether the provenance is "plausible", that is whether it is acceptable or not acceptable as the processes documentation [5]. If the hash/signatures are not consistent, for example *Process Executor 1* updates the provenance and data, the auditor can only decide that the provenance is not acceptable, but the auditor cannot decide which process executor is honest.

Even if *Process Executor 2* keeps the hash/signature of the provenance and data created by *Process Executor 1*, if *Process Executor 1* updates the provenance and data, the hash/signature cannot prove the claim without the existence of the original provenance and data. Hash/signature chain also cannot detect updates to the latest provenance and data. A malicious process executor can also abuse the provenance by simply deleting the provenance and data.

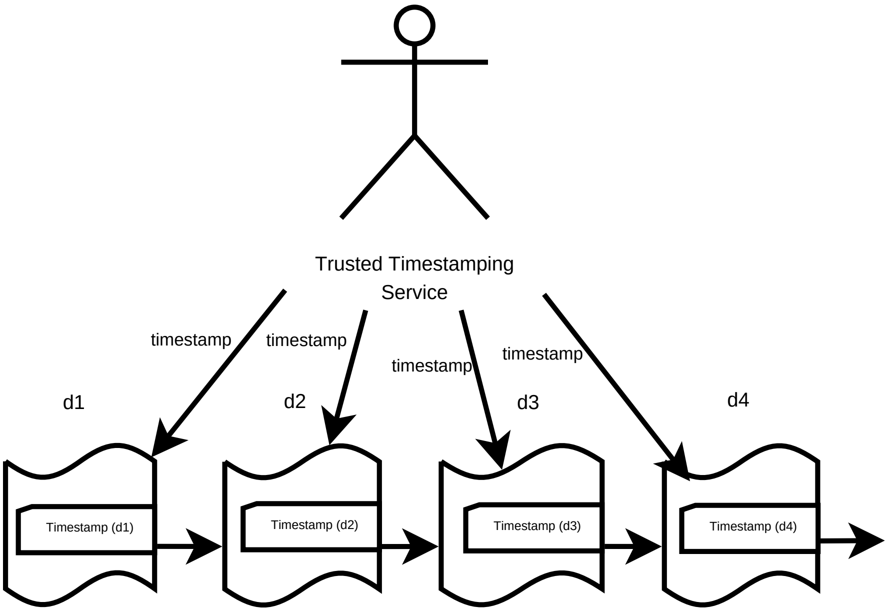

Figure 3.5: Trusted Timestamping Service

Another method is by using a secure timestamping service. Gadelha et al. implement a simple time-stamp mechanism for protecting the provenance [108] in a grid system. In their scheme, the provenance is signed by the data owner and hash of the signature is sent to a Time-Stamp Authority (TSA) that appends a timestamp to the hash. The hash and the timestamp is signed by the TSA and send them back as a provenance record receipt. This scheme can prevent repudiation, but it can not detect a deletion and update to the provenance. A data owner can simply delete the provenance without being detected or update the data by requesting new correct timestamp. The timestamp cannot also prove the relationship between the data other than proving the a data is created after another (by checking their creation times in the timestamps).

3.3 Preliminaries

3.3.1 Modeling the Security of the Provenance

In the integrity attack model, we are concerned to the type of attacks to the provenance by updating/deleting the provenance assertions. We define the integrity problem of the provenance by defining the attacks into two attack models: (1) inconsistent claim attack, (2) inconsistent interpretation attack. An inconsistent claim attack is an attack that cause the claims between at least two process executors do not match each other, and the auditor cannot decide which one is honest. An inconsistent interpretation attack cause different interpretation between the different auditor that access the provenance at different times.

In the Oxford dictionary, consistent is defined as "acting or done in the same way over time, especially so as to be fair or accurate" [27]. It is very important to keep the consistency of the provenance, otherwise the judgment made by the auditor can be invalid over time. As a history record, many researchers argue that the provenance should be immutable (no change is allowed in the history records) [4, 18, 32, 33, 44, 48, 109] the provenance represents the facts in the past.

We describe the consistency problem in the provenance by using the following scenario: An auditor ADT_1 checks the provenance at time t_1 and make a decision. Another auditor ADT_2 also checks the provenance at time t_2, and makes a decision. A consistent provenance should provide the same view for either ADT_1 and ADT_2.

Definition 3.1. The provenance is consistent if the auditors ADT_1 and ADT_2 have a consistent view of the provenance on any different times.

We show that the consistent provenance should be immutable. If we allow any updates, the auditor will get different interpretation over time.

Theorem 3.2. *A consistent provenance should be immutable.*

Proof. The proof by contradiction: if the provenance is mutable, and we allow changes, then the provenance can have different representation at different time, which is by definition is not consistent. □

We model the inconsistent claims attack as a game played by the attacker and the auditor. In the game, the challenger is the auditor needs to detect whether the provenance assertions are consistent or not. Before the game, the auditor may not have access to the provenance assertions. We define the attacker as one or more malicious process executors who want to avoid responsibilities for the processes. The attackers can access, update and delete all provenance assertions and data that were created by them. The attacker has access to a signature service for one of the process executors. The game is described in the Algorithm 5.

The inconsistent interpretation claims attack has a slight different characteristic where the attacker has access to the signing oracle of all process executors (described in Algorithm 5).

3.3.2 Definition of the Provenance

We define the provenance as a graph that can be recorded as a collection of assertions. A provenance of process that takes a collection of inputs $\{I_i\}$, produces an output O, executed by process executor identified by Cid is stored in a database PS in the forms of assertion $PAsrt$ as follows:

$$\begin{aligned} PAsrt =& a_i | r_i \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \end{aligned}$$

where:
Cid = the ID of the process executor
Pid = the ID of the provenance node
A = assertion about process execution
$\mathsf{ref}(I_i)$ = a reference to an input of the process
$\mathsf{ref}(O)$ = a reference to the output of the process
Cid'_i = the ID of the process executor that produce the input $\mathsf{ref}(I_i)$
Pid'_i = the ID of the provenance of the process that produce the input $\mathsf{ref}(I_i)$

The provenance is the collection of $PAsrt$ for a set of data outputs. The collection of $PAsrt$ can be depicted as a graph where the processes represent the nodes and the references to the other processes represent the edges.

Algorithm 5: Inconsistent Claims—Interpretation Attacks Game

```
$D, P, \sigma \leftarrow GetAll()$/* Get all data and provenance */
$Attack(D, P, \sigma, C_k|C)$/* Inconsistent Claims: Attacker has access to only one
process executor signing oracle, Inconsistent Interpretation: Attacker has
access to all process executor signing oracle */
return $Ver(D, P, \sigma, C)$
function $Attack(D, P, \sigma, C_k|C)$
  loop
     Choose any $d_i, p_i, \sigma_j$
     Create $d'_i, p'_i$
     $\sigma'_j \leftarrow SigningOracle(d'_i, p'_i, C_k)$
     $Update(d_i, d'_i, p_i, p'_i, \sigma_j, \sigma'_j)$
  end loop
end function
function $Update(d_i, d'_i, p_i, p'_i, \sigma_j, \sigma'_j)$
    $Delete(d_i, p_i, \sigma_j)$
    $Add(d'_i, p'_i, \sigma'_j)$
end function
function $GetAll()$
return $D, P, \sigma$
end function
function $Add(d_i, p_i, \sigma_j)$
   Insert $d_i$ to $DB$
   Insert $p_i|\sigma_j$ to $PS$
end function
function $Delete(d_i, p_i, \sigma_j)$
   Delete $d_i$ from $DB$
   Delete $p_i|\sigma_j$ from $PS$
end function
function $SigningOracle(d_i, p_i, C_k)$
   $\sigma_i \leftarrow sig_{C_k}(d_i, p_i)$
return $\sigma$
end function
function $Ver(P, D, \sigma, C)$
   /* Check the consistency of the provenance */
return 0|1 /* return 1 if the provenance is consistent, 0 otherwise */
end function
```

3.4 Proposed Scheme

3.4.1 Extended Hash/Signature Chain

We extend the hash/signature chain to allow the chaining to the graph form rather than the chain form originally the hash/signature chain is used. First, we need to keep the hash of the data in the provenance:

$$\begin{aligned} PAsrt =& a_i|r_i|Int(O) \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \\ Int(O) =& \mathbf{Hash(O)} \end{aligned}$$

To implement the extended hash/signature chain we need to also keep the chain to the parent nodes. So, we record each provenance assertion $PAsrt$ in the following format ($RCert$ is certificate released by the Trusted Counter Server (TCS), we will describe $RCert$ in the following section):

$$\begin{aligned} PAsrt =& a_i|r_i|Int(O)|Int(r_i) \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \\ Int(O) =& \mathbf{Hash(O)} \\ Int(r_i) =& \{\mathbf{Sign}_{Cid'}\mathbf{(PAsrt'), RCert'}\} \end{aligned}$$

The complete provenance assertion $SPAsrt$ is recorded as follows:

$$\begin{aligned} SPAsrt =& PAsrt, Int(PAsrt) \\ PAsrt =& a_i|r_i|Int(O)|Int(r_i) \\ Int(PAsrt) =& \mathbf{Sign}_{Cid}\mathbf{(PAsrt), RCert} \end{aligned}$$

3.4.2 Labeling Each Assertion with Unique Counter

We need to have a trusted entity called the Trusted Counter Server (TCS). The TCS receives inputs the group number G and $Sign_{Cid}(PAsrt)$ that is a signature on $PAsrt$ by the process executor Cid.

For a group number G of the provenance nodes, the TCS assign a unique counter number $R \in \{1, 2, ...\}$ to each provenance node and keeps the latest counter number N for the group in a trusted storage (see Figure 3.6).

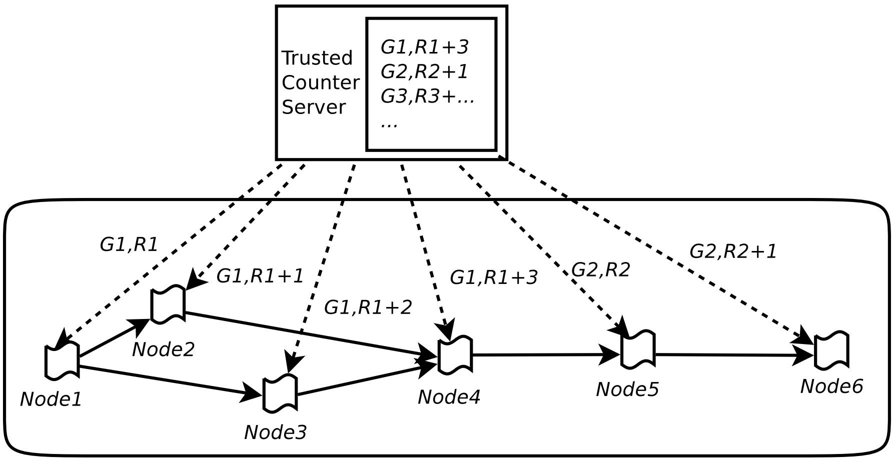

FIGURE 3.6: Trusted Counter Server (TCS)

The certificate issued by the TCS for a provenance assertion $PAsrt$ created by process executor Cid is defined as follows:

$$RCert = \langle G, R, \mathsf{Sign}_{TCS}(\mathsf{Sign}_{Cid}(PAsrt), G, R)\rangle \tag{3.1}$$

3.4.3 Secure Provenance Recording Protocol

In the secure provenance recording protocol, for each record, we should ask the counter to the TCS. An implementation is by employing the Provenance Store Interface to request the TCS for each assertion submitted by the process executor.

Figure 3.7 shows the parties that are involved in the provenance recording and auditing. We include a trusted party the Trusted Counter Server whose task is to proved the trusted counter for each provenance assertion.

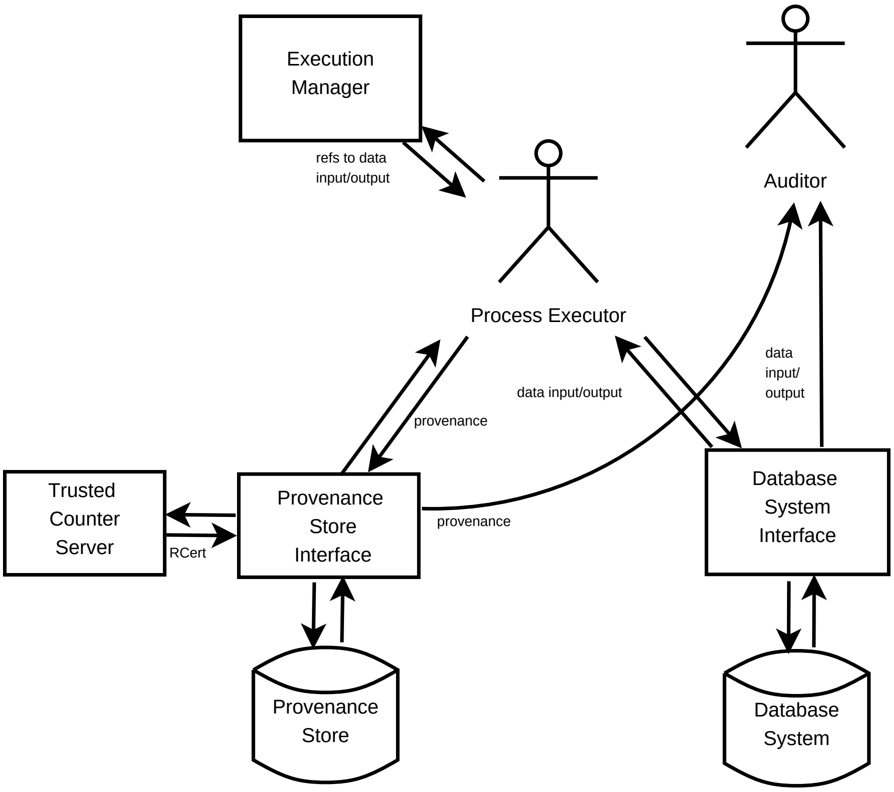

FIGURE 3.7: A Model of Secure Provenance System

Below, we describe the steps that are executed by the parties in a session of secure provenance recording. The protocol consists of five groups of steps as follows:

1. the Execution Manager invokes the process by providing references to inputs $\{\mathsf{ref}(I_i)\}$. The process executor retrieves the inputs from DB through its interface DBI.

$$EM \rightarrow C \quad : \quad \mathsf{Execute}(Qid, \{\mathsf{ref}(I_i)\}, \{Pid_i'\})$$

2. The Process Executor retrieves the inputs from DB through its interface DBI. The Process Executor executes the process stores the output O to the DB and sends back the reference of the output ($\mathsf{ref}(O)$) to the Execution Manager.

$$\begin{aligned}
C \rightarrow PSI \rightarrow PS &: \mathsf{Retrieve}(\{\mathsf{ref}(I_i)\}, \{Pid'_i\}) \\
PS \rightarrow PSI \rightarrow C &: \langle \mathsf{r_i} | \mathsf{Int}(\mathsf{r_i}) \rangle | \mathsf{false} \\
C \rightarrow DBI \rightarrow DB &: \{\mathsf{ref}(I_i)\} \\
DB \rightarrow DBI \rightarrow C &: \{I_i\} \\
C \rightarrow DBI \rightarrow DB &: \mathsf{ref}(O), O \\
DB \rightarrow DBI \rightarrow C &: \mathsf{success}|\mathsf{fail}
\end{aligned}$$

3. Provenance Recording
   The Process Executor creates the provenance assertion $PAsrt$ and sends to PSI.

$$C \rightarrow PSI \rightarrow PS \quad : \quad \mathsf{SubmitPAsrt}(PAsrt, Sign_{Cid}(PAsrt))$$

4. The PSI checks the signature $\mathsf{Sign}_{Cid}(PAsrt)$, whether it has been submitted before to prevent replay attack. The PSI asks $RCert$ that consist of group number G and counter R that is increased by one for each group to the Trusted Counter Server.

$$\begin{aligned}
PSI \rightarrow TCS &: G, \mathsf{Sign}_{Cid}(PAsrt) \\
TCS \rightarrow PSI &: RCert = G, R, \mathsf{Sign}_{TCS}(\mathsf{Sign}_{Cid}(PAsrt), G, R)
\end{aligned}$$

$$\begin{aligned}
SPAsrt =& PAsrt, Int(PAsrt) \\
Int(PAsrt) =& Sign_{Cid}(PAsrt), RCert
\end{aligned}$$

5. The PSI submits $SPAsrt$ to PS. The Process Executor reports to the Execution Manager whether the whole process is successful or not.

$$\begin{aligned} PSI \rightarrow PS &: \mathsf{Submit}(SPAsrt) \\ PS \rightarrow PSI \rightarrow C &: \mathsf{success}|\mathsf{fail} \\ C \rightarrow EM &: \mathsf{Report}(Qid, \mathsf{ref}(O), Cid, Pid, \mathsf{success}|\mathsf{fail}) \end{aligned}$$

3.5 Security Analysis

We need to prove that this scheme is resistant to inconsistent claims and inconsistent interpretation attacks.

Theorem 3.3. *A set provenance of the assertions in a group G is consistent, that is no successful inconsistent claims and interpretation attack, if the number of the assertions is R and no missing counter.*

Proof. Proof by contradiction, assume that the consistent provenance in a group G consists of n provenance assertions $P = \{P_0, P_1, ..., P_{n-1}\}$. To do an inconsistent claim or an inconsistent interpretation attack, in the inconsistent claim/interpretation attack game, the attacker should create a set of provenance assertions $P' = \{P'_0, P'_1, ..., P'_{n-1}\}$ where at least one of them has been updated using the same counter. Because the counter for a group cannot be generated more than one times, it is a contradiction. □

Theorem 3.4. *By using the above scheme, if the signature scheme* Sign *is secure and TCS does not collude with any other parties, the auditor can always detect any changes to the provenance nodes.*

Proof. We should show that any changes and deletions to the provenance assertions can be detected. We show that it is not possible to have a consistent provenance assertions after any alteration, or deletion of the provenance graph:

1. To alter the content of $PAsrt$ consistently, an attacker should also update $Int(PAsrt)$. If the process executor Cid corrupts and re-creates the signature $\mathsf{Sign}_{Cid}(PAsrt)$, the alteration can be detected from $RCert$ on $Int(PAsrt)$. The process executor can submit new provenance to get new correct R from the TCS and deletes the previous one, but the TCS will give

a new number R so that the total number of nodes will be less than the number recorded by the TCS.

2. To delete a node consistently, deletion causes the number of nodes to decrease, so that the total number of nodes is not the same as N.

□

Corollary 3.5. *By using the above scheme (described in the section 3.4.3), if all provenance assertions $PAsrt$ in group G is consistent, for each node whose counter $R \neq 1$ and $R \neq N$, there are two other nodes submitted by PO whose counters are $R + 1$ and $R - 1$.*

Proof. Because an assertion cannot be deleted without being detected, each assertion has a unique counter number occupying all numbers from 1 to N where N is the number of nodes in group G. So that for any number of counter $R \neq 1$ (counter for the first node) and $R \neq N$ (counter the last node) there will be other nodes whose counters are $R + 1$ and $R - 1$. □

3.6 Storage Requirements for the Integrity Schemes

Each assertion stores:

$$\begin{aligned} SPAsrt =& PAsrt, Int(PAsrt) \\ PAsrt =& a_i | r_i | Int(O) | Int(r_i) \\ Int(PAsrt) =& Sign_{Cid}(PAsrt), RCert \end{aligned}$$

where

$$
\begin{aligned}
PAsrt =& a_i | r_i | Int(O) | Int(r_i)\\
a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle\\
r_i =& \{Cid'_i, Pid'_i\}\\
Int(O) =& Hash(O)\\
Int(r_i) =& \{Sign_{Cid'}(PAsrt'), RCert'\}
\end{aligned}
$$

The size of $Int(O)$ and $Int(PAsrt)$ are constant for each assertion, while the size of $Int(r_i)$ depends on the number of inputs. For n number of inputs, each node needs additional storage for the integrity scheme $\mathsf{Size}(Int(O)) + \mathsf{Size}(Int(PAsrt)) + n(\mathsf{Size}(Int(r_i)))$. If the size of $Int(O) = \mathsf{Size}(Hash)$, the size of $Int(PAsrt) = \mathsf{Size}(Sign) + \mathsf{Size}(RCert)$, the storage for the integrity scheme is $\mathsf{Size}(Hash) + (n+1)(\mathsf{Size}(Sign) + \mathsf{Size}(RCert))$.

3.7 Beyond the Counter: Certificate of Relationships

The malicious process executors may remove some or all parts of the nodes to avoid the responsibility to the deleted nodes. Removing all nodes is a vandalism that can not be solved without storing all provenance in a trusted storage. Removing some parts of the provenance assertions cause problems to the other (including the honest) process executors we call "indirect relationships" problem.

For example, *Process Executor 3* uses an output produced by *Process Executor 2*, while *Process Executor 2* uses an output produced by *Process Executor 1*, so that *Process Executor 3* has indirect relationship to *Process Executor 1*. Rather than changing the provenance, *Process Executor 2* may also delete the provenance to avoid responsibility. If an inhonest *Process Executor 2* deletes its provenance, the auditor cannot confirm the indirect relationship between *Process Executor 3* and *Process Executor 1* even if both of them are honest. By having the counter, we can extend the signature chain to record more relationship by having the certificate of the relationships.

3.7.1 Including Certificate of Relationships

We can improve the resiliency of the counter mechanism by including the certificate of the relationships as follows. The TCS issues a u bits certificate S for each provenance nodes in the group G that confirms the relationships with the parents with counter numbers $R-u, R-u+1, ..., R-1$. For u bits of certificate S, the value of the bit $(R-u+n)$th bit is 1 if it has relationships with the node number $R-u+n$ for $0 \leq n < u$, otherwise its value is 0.

For example, a node with the counter number 9 has parents nodes with the counter numbers 6 and 7. The node number 6 has parents nodes numbers 3 and 4, and the node number 7 has parents nodes numbers 5. If we use a 4 bits certificate of relationships (in implementation, we use larger bits for example 64 or 128 bits), the TCS will assign 0110 to node number 6. Because for the node with $R = 6$, we record the relationships to nodes numbers $6-4+0, 6-4+1, 6-4+2, 6-1 = 2, 3, 4, 5$ in S. In this case the value of S is 0110, it means that node with number 6 has direct/indirect relationships with nodes number 3 and 4. The node with counter number 7 is assigned $u = 0010$, because S records the relationships with the nodes numbers $3, 4, 5, 6$ and it has relationships with the node number 5.

The TCS assigns S for nodes with counter number 9 to record the relationships of node number 9 with nodes number $5, 6, 7$, and 8. The TCS assigns value S for node number 9 by combining all values of S on the parents (the nodes with number 6 and 7). First, the TCS records S by checking the relationship with the parent. Because the parents are 6 and 7, the value of $S = 0110$. Then, the TCS computes the value of S in each parent by left sifting the S at the parent $R - R'$ bits where R' is the counter on the parent, for example for the parent 7 whose $S = 0010$, the TCS shift left $9-7 = 2$ bits to get 1000, and for the parent 6 whose $S = 0110$, the TCS shift left $9-6 = 3$ bits to get 0000. Finally, the TCS combines them with a binary OR operation to get $\mathsf{OR}(0110, 1000, 0000) = 1110$.

We update the original $RCert$ as follows:

$$RCert = \langle G, R, S, \mathsf{Sign}_{TCS}(\mathsf{Sign}_{Cid}(PAsrt), G, R, S)\rangle$$

3.7.2 How to recover the missing nodes

By using the certificate of the relationships, the auditor can detect any deletion to the provenance graph. If there are some missing nodes, the auditor still be able to decide the causal relationships between some nodes by checking the certificate of relationships issued by the TCS. The certificate of relationships include u bits S that record the index $R - u, R - u + 1, ..., R$ for each node to record the causal relationships to u numbers of the previous nodes. After a deletion, the auditor can recover indirect relationships between existing nodes. For a better recoverability, we need to use large size of S.

We should know that a node is updated from the missing counter, however we cannot detect which one is deleted. There are some cases:

1. Check each nodes whose counter larger than the missing nodes and check the relationship in the node. If we find the missing counter in the certificate of relationship then FINISH. In this case, the provenance is still usable, and we have the possibility to recover.

2. If the missing counter is not found, the missing node should include the leaf nodes. It renders the provenance as INVALID. The provenance cannot be recovered.

3.8 Discussion: Public Key Infrastructure and Replay Attack

To implement the integrity scheme, we need to assume that each Process Executor, the PSI and the TCS have a pair of public key and private key, and each party can retrieve the public keys certificates of the other parties securely. For example in the case of application of provenance in a hospital, each actor (i.e., a physician) should have a pair of public key and the private key. They can also access the public key certificates (to access the public keys) of all other parties securely. We believe this assumption is acceptable because of common usage of the public key system, for example the Public Key Infrastructure (PKI) or alternatively decentralized trust management with the web of trust in PGP [110, 111].

For the integrity checking, in the TCS we can store information other than the number of nodes, for example the list of the provenance nodes and also hashes/signatures of all nodes. However, this alternative has some drawbacks. The first is we need more storage to store the information because for each node the TCS stores the hashes and signatures. The second drawback is in integrity checking, the auditor should download all of the integrity data from the TCS while using our method, the auditor only needs to ask the TCS one time to ask the number N (to check a deletion in integrity checking no. 3). The last drawback is the security mechanism depends only on the TCS while in our model, the security mechanism is distributed among many parties: the Process Executor, the PSI and the TCS.

We also assume that a replay attack can be prevented by checking signature for specific Cid. If the signature has been stored in the provenance store, the PSI simply ignore the request.

3.9 Performance Analysis

In Appendix A, our experimental results show the feasibility of implementing our scheme in the real systems. Our experimental results show that the process executors need almost constant execution times (with small growth) to execute the hash and sign functions, while the provenance store interface needs similar (almost constant) execution times. Our experimental results also show that the TCS needs constant execution times regardless the size of the provenance node. Most of the execution times that are needed to submit the provenance nodes to the provenance store are the times to send the nodes through the network.

3.10 Conclusion

In this chapter, we have discussed our proposed method to protect the integrity of the provenance from the inconsistent claims and interpretations attacks. Our method combines the hash, the signature chain and also employing a Trusted Counter Server (TCS) to prevent the malicious attacker easily changes the provenance (to do the inconsistent claims and interpretations attacks) without being detected. We also performed some real experimentals to measure the performance and show the feasibility of our method.

Chapter 4

An Access Control Model for the Provenance Recording System

4.1 Introduction

Provenance is a documentation that describes the processes to produce the data. The provenance is recorded in the form of assertions created by the process executors. The parties who have access to provenance assertions may infers some information about the data. If the data is sensitive, the provenance of data may be also be sensitive. But it is possible, the data is sensitive while the provenance is not or vice versa.

For example, in the case of the letter of recommendation for application to universities, the data is sensitive while the provenance is not, because the student who is recommended in the letter of recommendation does know the people who wrote the letter of recommendation, but the student is not allowed to open the envelope of the letter and read the letter [44]. In an employee's performance review, the provenance is sensitive while the data is not, because the employees are encouraged to read the data produced by the employee's review process, but they are usually not told who had inputs in writing of the process.

To restrict the accesses to the sensitive provenance, the provenance system should implement an access control mechanism. A simple approach to implement the access control is by creating a program that checks access by any parties to each provenance assertion and define policy for each combination of the party and the assertion. The access control program should store all information and access definition about each assertion and relationships between assertions. Using this method, we can implement almost all policies. However, because there is no structure of the access, the administrator should decide the access policy manually for each provenance assertion. It means that the security administrator should analyze the policy of all combinations of the parties who need to access the provenance (i.e., the auditors) and the provenance assertions and define the policy for all combinations. For a large numbers of the parties and the provenance assertions, the policy also need a large data space to store.

The access control models normally have frameworks that can be used to help the administrator to define the access control policy consistently. A consistent policy means that the policy applies a common rule for all policies defined in the system. For example, in the Role Based Access Control (RBAC), each user should be assigned to *roles* [112]. A group of access rights are assigned to *permissions*. Access policies are mapping of the roles to the *permissions*. The access control model can also include the security model to decide the policy. For example, in the LaPadula Model, a lower security level of users cannot access access a higher security level of objects cite.

The main principle that should be followed by the access control policy is the minimum access policy principle. In this principle, the access is only granted to the party that really needs the access to do his/her job. The access that is not related to his/her job should be denied.

4.1.1 The Problem of Access Control to the Provenance

The main purpose of an access control system is to restrict the access by the auditors to the provenance. The access control to the provenance has different characteristics to the access control to the regular data.

In the provenance, the basic information is the assertion about process and origin of data. When an auditor accesses the provenance, he/she may access information about process to produce the data, the process executor, and also references to the origin/source of the data. These information may be sensitive because it describes the process to produce sensitive data or because the sources of the data are sensitive. So, the access control system to the provenance should be able to restrict accesses by considering the sensitivity of information about the process and the origin/source of data.

When an auditor needs to check the processes that led to a data object, ideally, for a full traceability, the auditor needs to have access to the process and all direct/indirect origins. The access control system should be able to decide whether the auditor is granted to trace all of the origin/source of the data. It is possible that the access control policy states that parts of them are sensitive to specific auditor. So, the access control system to the provenance should be able to define the policies and implement access control to provenance by considering the direct and indirect relationships in the provenance assertions.

4.1.2 Contributions

In this chapter, we propose a method to implement access control system to the provenance by considering the sensitivity of information about the process and the origin/source of data and also whether the auditor is allowed to access all direct/indirect relationships. We define the access right we call TRACE, that is rather than controlling access for each provenance assertion, we implement the access collectively to the assertion that describes a data object and all assertions that describe the origin of the data object. We combine TRACE with the multi-label method for efficient access control definition to support more granularity of the access restrictions. We propose the implementation by using trusted reference monitor and also alternative implementation by using encryption.

4.2 Related Work

Braun et al. have discussed security issues on provenance [44] although they did not propose any security system related to the issues. They identified some of the security characteristics of provenance. The first is that provenance differs from data in that it forms a directed acyclic graph (DAG), so we need to have a security model for a directed acyclic graph (DAG). The second issue is that sensitivity level of data and its associated provenance may be different. It is possible that the provenance be more sensitive than data or vice versa. They also show some situations where the provenance information has different sensitivity level from the data documented by the provenance [44]. They give an example of employee's performance review where the employees are encouraged to read the data produced by the employee's review process, but they are usually not told who had inputs in writing of the process. In this case, the sensitivity level of provenance is higher that sensitivity level of the data. Another example is the letter of recommendation, e.g., letter of recommendation for application to universities. In this case, the student who is recommended in the letter of recommendation does know the people who wrote the but the student is not allowed to open the envelope of the letter and read the letter. In this case, the sensitivity level of provenance is lower than the sensitivity level of the data.

Braun et al. also argue that provenance needs a new security model [44]. They also propose a security model for provenance based on observation of the usage of provenance [113]. They focus on the security model but do not deeply discuss how to protect integrity of the provenance. Their main proposal is that we need to control access to heads and tails of the edges and the attributes of the nodes in the provenance graph. However, there is no mechanism proposed to implement their access control model.

Tan et al. list six security issues in an SOA-Based provenance system [114]. These security issues are (1) enforcing access control over process documentation, (2) trust framework for actors and provenance stores, (3) accountability and liability for *p*-assertions, (4) sensitivity of information in *p*-assertions, (5) long-term storage of *p*-assertions, and (6) creating authorizations for new *p*-assertions. They emphasis that the first issue is unique to the provenance purposes because the requirements are different from regular data.

Groth et al. have proposed an architecture of provenance system including the security architecture in an EU sponsored project [2]. They have implemented the architecture in an SOA-based provenance store. They suggested that access control should be specifiable at the level of individual *p*-assertions and at individual elements within *p*-assertion if needed. They also suggested to use role-based access control and content-based access control although no detail explanation and implementation of their proposal.

Syalim et al. discusses the grouping method for improving efficient of access control to provenance graph [104]. They analyze the efficiency of the access control to the provenance that employs grouping mechanism. Chebotko et al. [115] proposed a secure scientific workflow provenance querying with security view. Security view is a subset of data and processes. Another related work is the work Nagappan et al. [116]. They proposed a model of sharing confidential provenance information where an actor who are willing to share the provenance information can share the query for that provenance information.

Ni et al. proposed an access control language for a general provenance model [117]. They define the provenance of a piece of data as "the documentation of messages, operations, actors, preferences, and context that let to that piece of data". Operation is a process performed on or caused by some messages. Actors can be applications or human being. Context and preferences are special messages that are used in operation.

Ni et al. proposed access control language for provenance although no specific implementation is described [117]. The access control language consists of the components as follows:

- *Target.* The target includes the subjects and records to which the policy is applied. The subject can be a collection of users, the records can be also a collection of provenance records. We can include restriction for specific selection of the subjects in the collection.
- *Condition.* Condition is a boolean expression that define more requirement for the access, for example time limit, location, etc.
- *Effect.* Effect is the rights of the subject to the provenance records. The effect supports the following values: Absolute Permit, Deny, Necessary Permit, and Finalizing Permit.
- *Obligation.* An obligation is an operation that should be executed by the subject before (pre-obligation) or after (post-obligation) accessing the provenance records. For example, the actors who own provenance record require each user provide their agreement before access, or to be informed after access.

Cadenhead et al. proposed [1] access control method by query pattern to the provenance in the form of XML using SPARQL. They use Open Provenance Model and propose access control language shown in Figure 4.1. The access control can decide the access by 5 relationships defined in the OPM model, but no definition for grouping the access control based on a deeper relationships.

```
<policy ID="1" >
  <target>
    <subject>anyuser</subject>
    <record>Doc1_2</record>
    <restriction>
      Doc1_2 [WasGeneratedBy] process AND
      process [WasControlledBy] physician|surgeon
    </restriction>
    <scope>non-transferable</scope>
  </target>
  <condition>purpose == research</condition>
  <effect>Permit</effect>
</policy>
```

Figure 4.1: An Access Control Language for Provenance [1]

4.3 Preliminaries

4.3.1 Definition of the Provenance

Our proposed scheme works for the provenance that is defined as a graph. We define the provenance of process that takes a collection of inputs $\{I_i\}$, produces an output O, executed by process executor identified by Cid is stored in a database PS in the forms of assertion $PAsrt = a_i|r_i$ where a_i is the node and r_i represents and edge, as follows:

$$\begin{aligned} PAsrt =& a_i|r_i \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \end{aligned}$$

where:

Cid = the ID of the process executor

Pid = the ID of the provenance node

A = assertion about process execution

$\mathsf{ref}(I_i)$ = a reference to an input of the process

$\mathsf{ref}(O)$ = a reference to the output of the process

Cid'_i = the ID of the process executor that produce the input $\mathsf{ref}(I_i)$

Pid'_i = the ID of the provenance of the process that produce the input $\mathsf{ref}(I_i)$

4.3.2 Provenance Storage

As described in the previous chapter, a secure provenance can be represented as $SPAsrt$ as follows:

$$\begin{aligned} SPAsrt =& PAsrt|Int(PAsrt) \\ PAsrt =& a_i|r_i|Int(O)|Int(r_i) \\ a_i =& \langle Cid, Pid, A, \{\mathsf{ref}(I_i)\}, \mathsf{ref}(O)\rangle \\ r_i =& \{Cid'_i, Pid'_i\} \\ Int(PAsrt) =& Sign_{Cid}(PAsrt), RCert \end{aligned}$$

We need to store each of the provenance assertion in a persistent storage, e.g., database that is accessible to the auditors. In the real implementations, we may use either relational databases or XML [117].

4.3.3 Access Control Enforcement

To implement the access control system, we need to assume the existence of the trusted reference monitor in Provenance Store Interface (PSI) that intercepts each access to the provenance. As described in the previous chapter, the parties in the provenance system are the following (depicted in the Figure 4.2):

1. The Process Executors
2. The Database System (DB) and Database System Interface (DBI)
3. The Provenance Store (PS) and Provenance Store Interface (PSI)
4. The Execution Manager
5. The Auditors (ADT)

The trusted reference monitor is implemented as an Access Control Module (see Figure 4.3). The Access Control Module in the PSI consists of two sub-modules as follows (similar to the ITU-T standard [118] that defines two functions – access

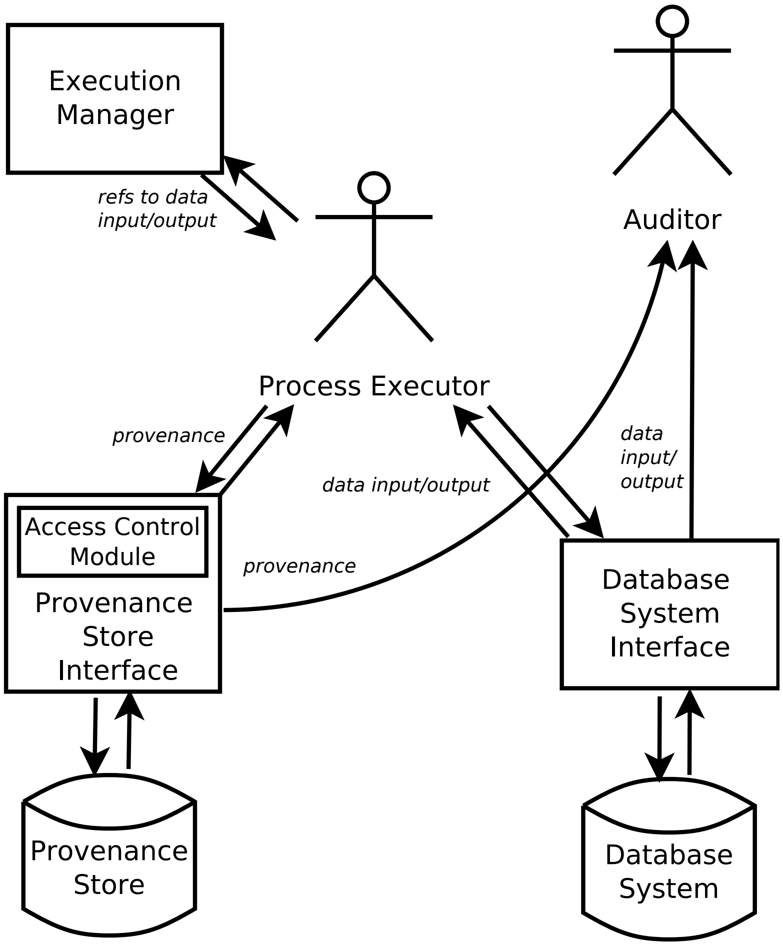

Figure 4.2: Participants in the Provenance System

control enforcement and access control decision functions – for a trusted reference monitor):

1. Access Control Enforcement (ACE) Module: This module enforces the access policy by intercepting/mediating access by user and ask Access Control Decision Module for the access policy

2. Access Control Decision (ACD) Module: This module decides which policy is applied for each access

The Access Control Module in the PSI intercepts accesses to the provenance through the Access Control Enforcement (ACE) module. The ACE module receives each access request to the provenance through PSI, and asks the Access Control Decision (ACD) module to decide whether to allow or deny the access.

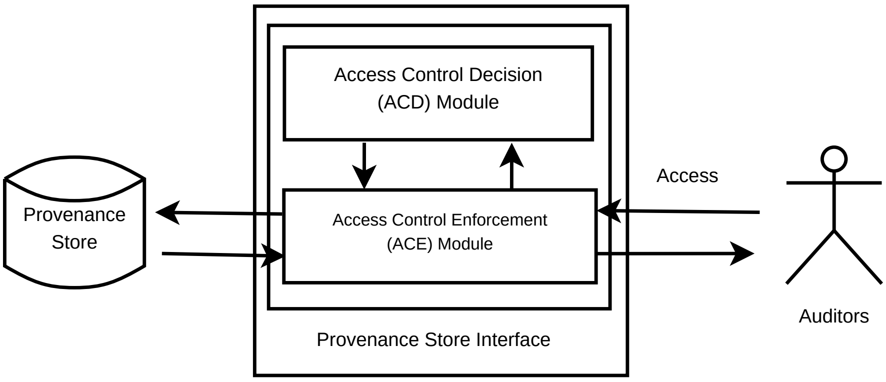

Figure 4.3: Access Control Module

The ACE module enforces the access decided by the ACD module by allowing or rejecting the access by the auditor. To implement the access control system, we also need to assume that the ACE module has authentication mechanism to check the identities of the auditors.

4.4 Proposed Access Control System

In this section, we describe our proposal of the access control system for the provenance. At first, we describe the security model we use in the access control system, then we describe the implementation of the security model, and the access control decision algorithm.

4.4.1 TRACE and Multilabels

The main intention of the auditor to access the provenance assertions is to evaluate the process and the origin of data. So that, the basic access right is accessing an assertion about the data and also tracing the origins/sources of the data. In our access control model, we define the access right to the provenance assertion as TRACE. An auditor who is given a TRACE is given rights to access an assertion and all of its ancestors (i.e., the origins) in the provenance graph model. Formally, we define the access right TRACE as follows:

Definition 4.1 (Definition of TRACE)**.** An access right TRACE $(ADT_i, SPAsrt_i)$ of an auditor ADT_i to an assertion $SPAsrt_i$ is defined as the right that is granted to auditor so that the auditor can access the assertion and also all ancestors of the assertion.

In our model, we define the access policy by using the TRACE and Multilabels that can be assigned for the each assertion and also each auditor. The access control decision for each assertion is computed by checking TRACE right and also by comparing the labels on the assertion and the auditor. We define the Multilabels as follows:

Definition 4.2 (Definition of Multilabels)**.** The Multilabels is a set $L \subseteq \{0, 1, ..., n-1\}$ for n number of labels where each member of L represents a label.

4.4.2 Implementation of the Multilabels

We record multilabel for each auditor and the set of assertions that define the TRACE access policy. Each auditor is assigned a set of labels (i.e., the multilabels). For each label, we record the information in the following format:

$$i|Description$$

where i is an integer, and *Description* is a string that describe the label. For example two labels of 10 and 25 with the descriptions “Financial Audit“ and “Process Audit” are recorded as follow:

10|Financial Audit
25|Process Audit

For each auditor, we record the information about the identity of the auditor and also the multilabels of the auditor in the following format:

$$\begin{aligned} ADT =& UserName|Id|TLabel \\ Id =& FullName|Organization \\ TLabel =& \{L_i\} \end{aligned}$$

Some examples of the record for the auditors are as follows:

$$john1|\text{John Doe}|\text{Google}|\{10, 11, 25\}$$
$$john2|\text{Johny Deep}|\text{Microsoft}|\{11, 25\}$$

For each provenance assertion that is identified by $\langle Cid, Pid\rangle$, we also need to record the Multilabels in the following format:

$$\langle Cid, Pid\rangle|TLabel$$
$$TLabel = \{L_i\}$$

Some examples of the access control policies are as follows:

$$\langle cut10, 201\rangle|\{10, 11, 25\}$$
$$\langle copy30, 409\rangle|\{11, 25\}$$

4.4.3 Implementation of the TRACE

TRACE is implemented by simply recording the pair of the auditor and the assertions. Some examples of the policies are as follows:

$$john1|\text{John Doe}|\langle cut10, 201\rangle$$
$$john2|\text{Johny Deep}|\langle copy30, 409\rangle$$

The above policies mean that the auditor *john*1 has TRACE right to the provenance assertion $\langle cut10, 201\rangle$ while the auditor *john*2 has the TRACE right to the provenance assertion $\langle copy30, 409\rangle$.

4.4.4 Access Control Decision

Each access control decision is computed by comparing the label of the auditor with the label of the assertions. However, not all assertions define the label. This is the main feature of our the access control for efficient and secure access. The auditor cannot access the assertions with empty label directly, but he/she may be able to access an assertion with empty labels by accessing a related assertion (i.e., the assertion that have causal relationship with the assertion with empty label).

We describe the access control decision as follows. To audit a data output d_i, the Auditor ADT needs to access the provenance assertion p_i. The auditor sends access query $\mathsf{Trace}(\mathsf{ref}(d_i))$ to PSI. PSI checks access policy and returns p_i and keeps all other assertions that are allowed to be accessed by the ADT in a temporary memory. During the session, the PSI decides the access by checking the temporary memory. The access control decision is described in the Algorithm 6.

As described in the Algorithm 6, to access a provenance of data output d_i, the auditor sends his/her identity and the information about the data output d_i. The ACE asks ACD whether this access is allowed or not. The ACD checks the label of the auditor (i.e., $ALabel$) and finds the provenance of d_i, that is p_i. The ACD gets the label of p_i (i.e., $PLabel$) and computes the intersection of $ALabel$ and $PLabel$. If the intersection is not NULL, the auditor is allowed to access p_i. Otherwise, the access is denied. So, for all assertion with empty label, the direct access is always denied.

If the access is allowed, the ACD traces the relationship of p_i by checking $r_i = \{Cid', Pid'\}$ (see Section 4.3.2), that is a set of the ids of p_i's parents. For each parent, if the label is empty the access is allowed, the ACD continues to the parent of the assertion with empty label (i.e., the grandparent of the original assertion). However, if the label is not empty, the ACD needs to compare (compute the intersection) the label with the label of the auditor. The access decision now depends on the intersection. If the intersection is not empty, the access is allowed and the checking is continued to the grandparent (like the access decision for a parent with an empty label). But, if the intersection is empty, the access is denied, and the ACD stops checking to the grandparent.

Algorithm 6: Access Control Decision Algorithm

```
Inputs: Auditor $UserName$, a data output $\mathsf{ref}(d_i)$
Outputs: Set of id of the provenance assertions $\{Cid, Pid\}$ or NULL
$ALabel \leftarrow GetALabel(Username)$ // Get all labels of the auditor
$\langle Cid, Pid\rangle \leftarrow GetPAsrt(\mathsf{ref}(d_i))$ // Get provenance assertion for $d_i$
$PList \leftarrow$NULL
if $\langle Cid, Pid\rangle \neq$NULL then
  $GetTrace(\langle Cid, Pid\rangle, ALabel, PList)$
else
  return NULL
end if
function $GetTrace(\langle Cid, Pid\rangle, ALabel, PList)$
$PLabel \leftarrow GetPLabel(\langle Cid, Pid\rangle)$ // Get all labels in $p_i$
if $(PLabel \cap ALabel) \neq$NULL then
  $Add(\langle Cid, Pid\rangle, PList)$
  $CheckLabel(\langle Cid, Pid\rangle, ALabel, Plist)$
else
  return NULL
end if
end function
function $CheckLabel(\langle Cid, Pid\rangle, ALabel, PList)$
$\{\langle Cid', Pid'\rangle\} \leftarrow GetParent(\langle Cid, Pid\rangle)$
if $(\{\langle Cid', Pid'\rangle\}$ =NULL then
  return NULL
else
  foreach $\{\langle Cid', Pid'\rangle\}$
  $PLabel \leftarrow GetPLabel(\langle Cid', Pid'\rangle)$
  if $(PLabel$ =NULL then
    $Add(\langle Cid', Pid'\rangle, PList)$
    $CheckLabel(\langle Cid', Pid'\rangle, ALabel, PList)$
  else if $(PLabel \cap ALabel) \neq$NULL then
    $Add(\langle Cid', Pid'\rangle, PList)$
    $CheckLabel(\langle Cid', Pid'\rangle, ALabel, PList)$
  else
    return NULL
  end if
  end foreach $\{\langle Cid', Pid'\rangle\}$
end if
end function
```

4.4.5 Security Analysis

We analyze the security by proving some facts as follows:

Theorem 4.3. *An auditor cannot access an assertion that defines no access right TRACE for the auditor.*

Proof. No trace for an auditor means that the assertion defines a trace policy but not applicable for an auditor. In the access control decision algorithm, the auditor is allowed the access in two following conditions:

1. the intersection of the trace label of the assertion and the auditor are not empty, so the trace policy is applicable for the auditor.
2. the trace label is empty and the assertions are the parent/grandparent (we can continue the relationship to all ancestors) of an applicable assertion.

From the above conditions, we conclude that the auditor cannot access an assertion that defines no trace in any conditions.

□

Corollary 4.4. *An auditor cannot access an assertion with empty label without having access to at least one assertion that has a relationship (i.e., as children or grandchildren) to the assertion with empty label.*

Proof. The proof follows from the proof of Theorem 4.3, that is from the second condition that allows access. □

From the above theorem and corollary, we conclude that the access decision come from two policy definitions: explicit and implicit policy. In explicit policy, the administrator should define the trace label, while in implicit policy the trace label of the assertion is empty but the assertion can be accessed by accessing the successor of the assertion. We should now proceed to the following theorem.

Theorem 4.5. *To enforce minimum access policy principle, if the provenance assertion is sensitive, the administrator should define the trace label for the assertion, while in the case of the provenance assertion is not sensitive the administrator can safely assign empty label to the assertion.*

Proof. In the minimum access policy principle, the auditor can only access the provenance that is needed to do the audit. By using this method, the system can enforce minimum access policy because the auditor can only access the assertions that have relationships to the accessible assertions. So the assertions that have no relationships or no applicable trace policy will be denied to be accessed by the auditor. Because to do the audit the auditor only needs to access an assertion and all assertions that have relationships to the assertions, the system effectively enforce the minimum access policy principle.

□

4.4.6 Performance Analysis

The access decision algorithm (the Algorithm 6) needs to decide which assertions can be accessed for each session. As shown in the Algorithm 6, the ACD needs to compute the intersection between the labels ($PLabel \cap ALabel$). If the access is allowed to a node, the ACD needs to check the parents of the assertion ($GetParent(\langle Cid, Pid \rangle)$) and if the label on the parent is not empty, the access control decision is (again) done by computing intersection between the labels ($PLabel \cap ALabel$).

If the label in a node is empty, the ACD simply stops and denies all accesses, no need to continue to the parent. The ACD needs to continue to the parents if and only if the auditor is allowed to access the node. So, for the best case, the access decision algorithm is stopped at the first node. As of the worst case, the access decision algorithm needs to check all ancestors of the node.

To improve the efficiency, the ACD can implement a lazy checking, that is by only deciding the access if the auditor requests the access to a node. In this case, the complexity of the access control decision algorithm (in the worst case) depends on the number of accesses that are needed by a typical auditor.

4.5 Alternative Implementation: Encryption-based Access Control

Encryption is an alternative to access control enforced in the Provenance Store. It is suitable in some situations, for example in the situation where the provenance store cannot be fully trusted (i.e., it resides in a cloud server) or in the situation where the database is highly vulnerable to attackers that break the OS and database access control.

In the encryption method, each sensitive data is encrypted with a key, and all authorized auditors should be provided with the keys for decrypting the data that he is authorized to access. The database server and any attackers breaking the provenance store cannot decrypt the sensitive data without having access to the decryption keys. The problem in the encryption method is how to manage a large number of encryption keys that should be provided to a large number of auditors with different access policy.

In this section, we show an encryption mechanism for provenance graph that allows tracing the process documentations. This access control is suitable in the provenance because an auditor normally needs to access all assertions that have the causal relationships to a specific assertions. This method also has advantages in key management because to grant access to an assertions and all of its ancestors, we only need to provide a few of decryption key. More specific policies are supported by a label-based access control.

4.5.1 Encryption Method

To enforce the trace-based access control, the provenance tuple (i.e., the assertion) is encrypted with two keys (KN and KL) by the Provenance Store Interface (PSI) before storing the provenance. Each provenance tuple is assigned two key generator values (nk and lk), which are not encrypted.

The encryption method is defined as follows:

$$
\begin{aligned}
EPAsrt =& \langle Cid, Pid, EP, \langle nk, lk \rangle \rangle \\
EP =& \mathsf{Enc}_{KL}(\mathsf{Enc}_{KN}(SPAsrt), \{KN'\})
\end{aligned}
$$

4.5.2 Key Generation

The Provenance Store Interface (PSI) keeps two master keys MKN and MKL in a trusted place. Keys KN and KL in each assertion are generated from MKN and MKL as follows:

$$
\begin{aligned}
KN &= \mathsf{Enc}_{MKN}(nk) \\
KL &= \mathsf{Enc}_{MKL}(lk)
\end{aligned}
$$

Key generator lk of an assertion identifies the label of the assertion (the assertions in the same labels have the same lk) while key generator nk should be unique for each assertion.

4.5.3 Provenance Recording Protocol

The assertions are encrypted by the Provenance Store Interface (PSI). To encrypt $SPAsrt$ using KN and KL to produce EP the PSI executes the following steps:

1. The PSI generates nk (which is unique for each assertion) and defines the label number lk (where the assertions in the same label have the same lk). For an empty label, lk is set to $null$.
2. The PSI generates KN and KL (the mechanism is shown in Section 4.5.2).
3. The PSI encrypts the assertion with the key KN to get $\mathsf{Enc}_{KN}(SPAsrt, \{KN'\})$, where $\{KN'\}$ is the set of KN of the parents.
4. The PSI re-encrypts the assertion with key KL to get $\mathsf{Enc}_{KL}(\mathsf{Enc}_{KN}(SPAsrt, \{KN'\}))$.

4.5.4 Accessing the Provenance

An auditor who needs to access the provenance, performs the following steps:

1. The auditor starts from a leaf assertion.
2. The auditor decrypts $\mathsf{Enc}_{KL}(\mathsf{Enc}_{KN}(SPAsrt, \{KN'\}))$ with key KL and KN to get $SPAsrt$ and $\{KN'\}$.
3. The auditor proceeds to the parent of the assertions, if the parent has empty label, he/she can decrypt the parent by using one of the keys in $\{KN'\}$, if the label is not empty, the auditor should have access to KL on the parent to be able to proceed.
4. The access is finish if either reaching a root assertion (the assertion that has no parent) or reaching an assertion with label that is not applicable (the auditor is not granted access by providing the label key KL)
5. If there is another leaf assertion, for the new leaf assertion start from the step 1.

4.5.5 Access Control Policy

In Figure 4.4, we show an access policy to the provenance that consists of seven assertions to four auditors that can be implemented using this encryption mechanism and also the TRACE method. In this policy, the assertions are divided into two labels: *confidential* and *unconfidential* where the members of *confidential* are $SPAsrt_4$ and $SPAsrt_6$. The others are members of *unconfidential.* All auditors can access the assertions that are members of *unconfidential* so the assertions do not need to be encrypted with a label key. The first auditor (no. 1 in Figure 4.4) can access $SPAsrt_6$ and its ancestors, but he/she cannot access the assertions that are member of *confidential* ($SPAsrt_6$ and $SPAsrt_4$). The second auditor (no. 2 in the Figure) can access the assertion $SPAsrt_7$ and its ancestors, and can also access all assertions that are members of *confidential.* The third auditor (shown in no. 3) can access $SPAsrt_4$ and all its ancestors and also assertions in *confidential.* The last auditor (no. 4) can access $SPAsrt_2$ and its ancestor, and he/she cannot access the assertions that are members of *confidential.* To implement this access policy we provide the four auditors encryption keys as follows:

- auditor 1: KN_6
- auditor 2: KN_7 and $KL_{confidential}$
- auditor 3: KN_4 and $KL_{confidential}$
- auditor 4: KN_2

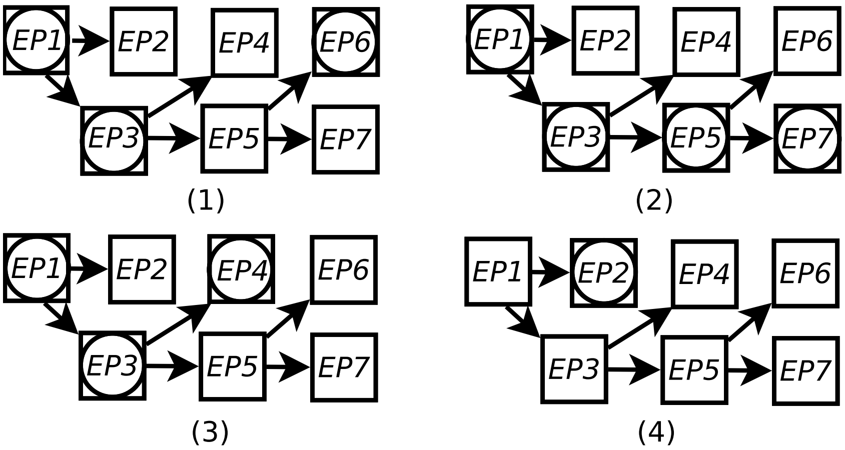

FIGURE 4.4: An example of the access control policy

As shown in the example above, to grant access to only an assertion and its ancestor, we need to provide the KN and KL keys, however, the trace is stopped when the auditor reaches the assertions whose label key KL are not applicable.

4.5.6 Performance Analysis

In Appendix A, we performed experiments to measure the computation costs of the encryption mechanisms. As shown in Table A.2, the complexity of the encryptions are linear (with small growth) to the size of the provenance nodes. This result is predictable because the complexity of AES encryption (that is used in the experiments) is also linear to the size of the data. As of the complexity to access the provenance nodes, it is also linear to the number of nodes that are accessed by the auditors.

4.6 Conclusion

In this chapter, we have proposed the access control mechanism that can be enforced in a provenance recording system. Our access control method supports the graph model of the provenance by allowing access policy definition to restrict the auditors tracing the relationships between the provenance nodes in the provenance graph. By combining the trace-based access control with multilabels method, we show that our access control method can be used to enforce the minimum access policy principle in the provenance recording systems. We also showed an alternative access control mechanism by using encryption for better security.

Chapter 5

A Signature Scheme for a Sequence of Digital Documents

5.1 Introduction

A digital signature scheme (i.e., an RSA signature), takes inputs of a document and a signer private key, creates a small sequence of bytes that can prove the document is indeed originated to the signer, by assuming nobody except the signer can generate the signature for the document [83]. Anybody who have access to the correct public key (which is normally available and can also be authenticated publicly) can verify the authenticity of the document. The standard security model for a digital signature scheme is the existential unforgeability where we prove that the problem to forge the signature can be reduced to a known difficult/intractable problem.

5.1.1 Problem Description

In this chapter, we are concerned about the method to sign a sequence of digital documents (created by a party) that consists of many distinct documents (i.e., a sequence of messages $M = \{m_1, m_2, \ldots, m_n\}$) that are created sequentially. The signature of the sequence of a digital documents should prove two facts about the documents: the authenticity of each member of the documents (i.e., whether m_i is the member of the sequence and truly created by the party), and the order of the documents in the sequence (i.e., whether m_i is included before or after m_j in M). We identify two main differences of signing this type of documents with the signature scheme for a single document:

1. A new member of the sequence can be added later after signing the current (existing) members, so that one of the requirements is the signature for the sequence should also be created sequentially.
2. During verification of the members, we cannot assume that we have access to all documents in the sequence. We can only assume that we have access to the documents that will be verified.

5.1.2 Usage of the Proposed Scheme

An example of the usage of the signature scheme is to sign a data stream. The signer cannot sign all data in the stream in one procedure, so we need to sign the data sequentially. A possible charateristics of the data stream is that part of the data of the stream (because of several reasons) may be inacessible in the future.

Another usage of the signature scheme is to sign a large message or a large sequence of messages where some parts of the messages may be confidential, so that during verification, we need to provide a limited access to each user. We cannot create one fix signature for all documents (by combining all documents into a bigger document) because to verify a part of the document we need to have access to all parts the document. Although the user cannot access all parts of the documents, the users still need to prove these two facts (the authenticity of each member of the documents, and the order of the documents in the sequence) for the accessible documents.

5.1.3 The Basic Method and Our Previous Attempts

The basic method and our first attempt to sign the documents with the above requirements is first by simply appending a consecutive counter to each member of the sequence and then signing each consecutive member of the sequence with a standard signature. Let *Sign* be a standard signature scheme (for example an RSA signature) with a signing key k and c_i be a counter that represents the order of the documents in the sequence, we can generate the signature on m_i as $\mathsf{Sign}_k(m_i \| c_i)$, so that the signatures on $M = \{m_1, m_2, \ldots, m_n\}$ be $\mathsf{Sign}_k(m_1 \| 1)$, $\mathsf{Sign}_k(m_2 \| 2)$, $\ldots$, $\mathsf{Sign}_k(m_n \| n)$. This method can prove each member and also the order of the members without having access to other members of the sequence. However, there are some disadvantages of this approach:

1. To sign many sequences, the signer needs to include another information (i.e a unique *id*) to identify each sequence, and includes the *id* along with the counter. The problem is, in practice, it is error-prone and not natural to keep the information of each sequence, rather than signing the sequence directly. The signer and the verifier should track the *id* of the sequence, otherwise, there are possibility the collision of the *id*. If the signature should be created in different systems, it is not easy to manage the *id* of each sequence.

2. The counter does not represent a "hard-proof" about the order of the members in the sequence, so that it is possible to sign the documents out-of-order by using out of order counters. For example, the signer may sign the first member, the third member and after then the second member. Without a trusted party who provide a correct consecutive counter, the signature cannot convince the verifier that the document signed as the order.

3. A rather minor disadvantage: we need to store a full signature for each member of the sequence. We cannot combine (i.e., by aggregating) the signatures without changing the characteristics of the signature: the possibility to verify a member without having access to the other members.

Our second attempt was by using the method similar to the scheme proposed in this chapter, but used the plain RSA as the primitive instead of the modified RSA with message recovery. However, the scheme could not be proved secure in Existential Unforgeability under Chosen Message Attack (EUF-CMA).

5.1.4 Contributions

In this chapter, we propose an alternative method to sign a set of digital documents where the signature scheme can prove two properties (the authenticity and the order of each member) of the documents without having access to all documents.

Our method is basically by employing signature-chaining and using a signature with message recovery as the primitive. Concretely, our contributions are:

1. We define a signature scheme for a sequence of digital document by including more features to the standard signature, so that it can be used to generate the signature sequentially and also verifying the order of the members in the sequence.
2. We propose a variant of the signature with message recovery originally proposed by Bellare et al.
3. We propose a signature scheme for the sequence of digital documents using a signature with message recovery as the primitive.
4. We propose the security model of the scheme in the form of Extended Existential Unforgeability under Chosen Message Attack (EEUF-CMA) and Order Unforgeability Under Chosen Message Attack (OUF-CMA).
5. We prove the security of the scheme in the random oracle model.

5.2 Related Work

5.2.1 Plain Signature

Digital signature is an active research topic in cryptography. The widely used digital signature scheme, the RSA signature scheme, was proposed by Rivest, Shamir and Adleman [83]. The other popular signatures scheme are Elgamal signature [90], DSA signature, and Schnorr signature [91]. Boneh et al. proposed a short signature by assuming the existence of bilinear maps [93].

We can apply the plain digital signature scheme (the signature for a single document) in the context of digital signature for a sequence of digital document. We can either sign each member of the documents, or combine all members of the sequence and signing the result. In the first method, for n number of the documents in the sequence, the signer needs to produce n signatures for the documents, while in the second method, the signer only produces one signature. The problem with the first method is, there is no way to prove the order of the documents in the sequence. The signature produced in the second method can be used to prove the order of the sequence. However, it has a major disadvantage: each time we include a new member, the signer should recreate the new signature which is not efficient in term of computation and not applicable if the previous documents are inacessible.

Another strategy to sign the sequence of the documents with a plain signature is by including a counter or timestamp in the signature [119]. Let $Sign$ be a standard signature, and c_i be a trusted counter or timestamp, the signature on m_i is $\sigma_i \leftarrow \mathsf{Sign}(m_i \| c_i)$. However, as described in Section 5.1.3, this method requires the existence of the trusted counter or timestamp server.

5.2.2 Signature Chain

The other possible methods to sign the sequence of the digital documents are by using the signature-chaining method. The signature chain ensures integrity of a sequence of document by storing the signature of previous document in the next document. The basic form of the signature chain is: let $Sign$ be a standard signature, then the signature chain for a sequence of messages $M = \{m_1, m_2, \ldots, m_n\}$ is a set of the signature $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ where $\sigma_i = Sign(m_i \| \sigma_{i-1})$.

The signature chain can be used to verify authenticity and the order of the documents in the sequence, however it has a main disadvantage where we need to keep the full signature for each document and to verify the order the documents, in the worst case, we need to have access to all members of the documents. The signature chain is an extension of the hash chain proposed in [105]. Recently, the signature chain is famous in the form of block-chain that is used to record transactions of a digital money (i.e., bitcoin) [65].

5.2.3 Signature Aggregate

Signature aggregate technique is used to combine many signature into one fix signature. Some aggregate signature schemes have a restriction about the messages where the signed messages should be distinct, the other schemes have a restriction about the signer where the same signer cannot signed the messages more than one times [120]. In our scheme, we are focusing on the method to sign a sequence of distinct messages by a specific signer.

The RSA signatures produced by a party can be aggregated/condensed into a succinct signature by simply computing the product of each individual signature. Bellare et al. use the aggregation method for fast verification of batch RSA. For the signatures $\sigma_1, \sigma_2, ...\sigma_t$ [95, 121], we compute the aggregate as follows:

$$\sigma = \prod_{i=1}^{t} \sigma_i \mod n$$

where $\sigma_i \equiv h_i^d \mod N$ and $h_i = H(m_i)$.

The short signature proposed in [93] can be easily aggregated by computing the product of the each signature [96] (exactly the same method as batch RSA). For the input n signatures $\sigma_1, \sigma_2, ..., \sigma_n$, compute the aggregate:

$$\sigma = \prod_{i=1}^{n} \sigma_i \in G_1$$

To verify the n messages $m_1, m_2, ..., m_n$, for a public key g_2^x check whether

$$e(\sigma, g_2) = \prod_{i=1}^{n} e(H(m_i), g_2^x)$$

We need to assume the existence of the bilinear maps $e : G_1 \times G_2 \rightarrow G_T$ where for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}_{|}$ and $e(u^a, v^b) = e(u, v)^{ab}$.

There are some significant research on the techniques to aggregate the signature with some more specific properties. The first property is the possibility to check the order of aggregating each signature as have been proposed in the form of sequential aggregate signature schemes [122–131]. Another interesting property is the history-free characteristics of the sequential aggregate signature, where aggregation algorithm does not need to have access all previous signed messages [132, 133]. However, during the verification, we still need to have access to all of the signed messages.

Recently, Gentry et al [134] classified three main methods for the sequential aggregate signature (SAS) schemes. These methods are (1) LMRS (the methods proposed in [122]), (2) Neven (proposed in [127]), and (3) BGR (proposed by Brogle et al. [128]). They also proposed a unified framework for the sequential aggregate signature scheme by using two functions (an ideal cipher π and claw-free trapdoor permutation $\mathcal{F}$), where the first function π is first called by taking inputs the previous signature (the aggregate so-far – σ_{i-1}) and also the concatenation of the outputs of permutation used to sign each previous message with all of the previous messages. The second step takes the outputs of the first function π and uses an inverse of the permutation generated by $\mathcal{F}$. They also proposed the randomized and the history-free variants of their scheme.

Hohenberger et al. discussed a synchronized aggregate signature where each party is provided with a message and a time period t. The party can only sign at most once for each t [135]. The uses the RSA function as the primitive with the RSA modulus $N = p \cdot q$ and using a keyed hash H_K to map the time period $t \in [1..T]$ to a specific value e_t and computes $E = \prod_{j=1}^{T} e_j \mod \phi(N)$. Their scheme exploits the characteristics of the multiplication of signed messages using the RSA with the same modulus.

The sequential aggregate signature, similar to the signature chain, can prove the authenticity and the order of the sequence and we only need to store one aggregate signature for all members. The main advantage is we can compress many signature into practically one signature. However, the disadvantage is, normally during the verification we need to have access to all of the documents in the sequence.

5.2.4 Signature with Message-Recovery

Signature with message-recovery is a signature scheme that can recover the original or a part of the signed message. The plain RSA without hash is an example of the message-recovery signature, however, the plain RSA is not resistant to EUF-CMA attacks. Bellare proposed the message recovery variant of the RSA-PSS [89]. The other proposed signatures with message recovery (for discrete log-based digital signature) can be found in the work of Nyberg et al. [136–138], and also the work of Miyaji [139], and Abe et al. [140].

5.3 Preliminaries

5.3.1 Definition of the Signature

The standard signature consists of three algorithms: **KeyGen** that produces the random keys; **Sign** that is used to generate the signature; and **Verify** that checks whether the signature is valid. We define the signature scheme for the sequence of digital documents by including functions that are used in the context of digital signature for a sequence of digital documents. These functions are: appending a new member, checking whether the signature is plausible (that is by having no access to the documents we should be able to verify whether the signature is created by a party), verifying a member without having access to all other members and also comparing the order of two members.

Formally, the signature scheme for a sequence of digital documents consists of seven algorithms as follows:

KeyGen. The key generation function that produces a random public and private keys pair $\{pk, sk\}$ of the signer.

Sign. This function takes a private key sk, a set of messages $M = \{m_1, ..., m_n\}$, and returns a new signature σ for the sequence $\{m_1, ..., m_n\}$.

Append. This function takes a private key sk, a new member m_n and the current signature σ for the sequence $\{m_1, ..., m_{n-1}\}$, and returns a new signature σ for the sequence $\{m_1, ..., m_n\}$.

IsPlausible. This function is used to check whether the signature is acceptable (created by the party) even without having access to all members of the sequence. The function takes a public key pk, the signature σ, and returns **TRUE** if the signature is acceptable, otherwise returns **FALSE**.

IsMember. This function takes a public key pk, a message m_j and the signature σ, and returns **TRUE** if $m_j \in \{m_1, ..., m_n\}$, otherwise returns **FALSE**.

IsLater. This function takes two messages m_j and m_k and the signature σ, and returns **TRUE** if $m_j \in \{m_1, ..., m_n\}$, $m_k \in \{m_1, ..., m_n\}$, and m_k is included after m_j, otherwise returns **FALSE**. If the signature is not plausible returns **REJECT**.

Verify. This function takes a public key pk, a set of message messages $M = \{m_1, ..., m_n\}$ and the latest signature σ, and returns **TRUE** if **IsMember**(pk, m_i, σ) returns **TRUE** for each $m_i \in \{m_1, ..., m_n\}$, otherwise return **FALSE**.

5.3.2 Security Model

A standard signature scheme should be resistant to existential unforgeability (EUF) attack where the attacker, in the chosen message attack (CMA) model, having access to some chosen pair of the message signature tries to fabricate a valid pair of the message signature that previously not yet queried [88]. The EUF under CMA (EUF-CMA) can be modeled as the game between the challenger B and the attacker A, where the attacker can request the pair of message-signature in the hope at the end of the game the attacker outputs a valid pair of message-signature that has not been requested before. A signature is said broken in existential forgery if an attacker can forge a signature for at least a message. The EUF-CMA model can be adopted to model the security of the aggregate signature as in [96].

As described in Section 5.1.1, the signature scheme for the sequence of digital documents should prove two facts (the authenticity of each document in the sequence, and also the order of the document in the sequence) while having two requirements: the possibility to update the signature sequentially, and the possibility to verify the signature by having access to the subset of the members in the sequence. We model the security of the signature for a the sequence of documents into two models as follows:

5.3.2.1 Extended Existential Unforgeability Under Chosen Message Attack (EEUF-CMA)

We extend the EUF-CMA model by including the possibility of forging signature under the conditions where some parts of the messages are inaccessible (however, we require at least an element of the messages is accessible so that we can prove the forging). We model the Extended Existential Unforgeability Under Chosen Message Attack (EEUF-CMA) as the game between the challenger B and the attacker A where the attacker may forge the signature for a member of the sequence while the other members are inacessible. So, the main difference of EEUF-CMA and the EUF-CMA is to forge the signature, the attacker does not need to forge a full sequence. The attacker only needs to show a member of the sequence, while the other members may be assumed to be inacessible to the verifier both to the attacker and the verifier.

The model can be described as follows: to attack the scheme, the attacker is provided with the signer public key, and the signer can request some set of sequence and its signature pairs (the attacker can also request one by one), and later the signer outputs a valid signature for at least a member of the sequence. The EEUF-CMA model consists of three parts as follows:

Setup. The challenger B runs algorithm KeyGen to obtain a public key pk and a private key sk. The adversary A is given pk.

Queries. Proceeding adaptively, the adversary A requests signatures on at most q_S for the sequence of messages of its choice $m_1, ..., m_{q_s} \in \{0,1\}^*$. The challenger B responds to each query with a signature $\sigma_i \leftarrow \mathsf{Append}(sk, \sigma_{i-1}, m_i)$.

Output. Eventually, the adversary outputs a pair $(m*, \sigma*)$ and wins the game if:

1. $(m*, \sigma*)$ is not any of the results of the previous requests $(m_1, \sigma_1), ..., (m_{q_s}, \sigma_{q_s})$, and
2. $\mathsf{IsPlausible}(pk, \sigma*) = \textbf{TRUE}$, and
3. $\mathsf{IsMember}(pk, m*, \sigma*) = \textbf{TRUE}$.

5.3.2.2 Order Unforgeability Under Chosen Message Attack (OUF-CMA)

In our proposed scheme, the signature is created sequentially for each new member based on the order of the members in the sequence. So, the order of the signing represents the order of the documents. Another class of attack is changing the order for the previously signed messages. We model the attack as follows: the attacker requests the pair of the signature and the sequence, and later the attacker produces a signature with different order than the existing one.

We define this kind of attack to the signature scheme in term of Order Unforgeability under Chosen Message Attack (OUF-CMA), where the attacker can request the ordered pair of message-signature, and later can prove inconsistency in the order. The attacker can find inconsistencies by forging the signature or simply rearranging the signature into another valid signature (in different order). The attack game is similar to the EUF-CMA, except that the attacker can also outputs the previous requested message-signature pairs. Formally, the OUF-CMA model consists of three parts as follows:

Setup. The challenger B runs algorithm KeyGen to obtain a public key pk and a private key sk. The adversary A is given pk.

Queries. Proceeding adaptively, the adversary A requests signatures on at most q_S messages of its choice $m_1, ..., m_{q_s} \in \{0,1\}^*$. The challenger B responds to each query with a signature $\sigma_i \leftarrow \mathsf{Append}(sk, \sigma_{i-1}, m_i)$.

Output. Eventually, the adversary outputs a tuple $(m_j*, m_k*, \sigma_a*, \sigma_b*)$ and wins the game if:

1. σ_b* is not any of the results of the previous requests $(m_1, \sigma_1), ..., (m_{q_s}, \sigma_{q_s})$, and
2. $\mathsf{IsPlausible}(pk, \sigma_a*) = \textbf{TRUE}$, and
3. $\mathsf{IsPlausible}(pk, \sigma_b*) = \textbf{TRUE}$, and
4. $\mathsf{IsLater}(pk, \sigma_a*, m_j*, m_k*) = \textbf{TRUE}$, and
5. $\mathsf{IsLater}(pk, \sigma_b*, m_k*, m_j*) = \textbf{TRUE}$.

5.3.3 Complexity Assumptions

5.3.3.1 Assumption about the Hardness of the RSA problem

The security of our scheme is based on the assumption about the hardness of the RSA problem. That is, given the RSA public key (N, e), and an element $y \in \mathbb{Z}_N^*$, compute x where $y = x^e \mod N$. More formally:

Definition 5.1 (RSA Assumption)**.** Let Π_n be a function that produces large random primes. For every non-uniform Probabilistic Polynomial Time (PTT) adversary A, there exists a negligible function μ such that for all $n \in \mathbb{N}$:

$$\Pr\begin{bmatrix} p, q \leftarrow \Pi_n; N \leftarrow pq; & \\ e \leftarrow \mathbb{Z}_{\phi(N)}^*; y \leftarrow \mathbb{Z}_N^* & : \quad x^e = y \mod N \\ x \leftarrow A(N, e, y) & \end{bmatrix} \leq \mu(n)$$

5.3.3.2 Assumption about the Hash Functions

Another important assumption we need to have in the security proofs of our scheme is about the difficulty to find two messages that have the same hash outputs (collision resistant). We define the advantage of the attacker to find the collision of outputs of the hash function in the following definition [70].

Definition 5.2 (Collision Resistant)**.** Let $H : K \times M \rightarrow Y$ be a hash function, the advantage of an adversary B to find the collision of the outputs of H is defined as follows:

$$\begin{aligned}\mathbf{Adv}_{H}^{coll}(B) \quad &= \Pr[K_i \overset{\$}{\leftarrow} K; (M, M') \overset{\$}{\leftarrow} B(K_i) : \\ &\quad (M \neq M') \wedge (H_{K_i}(M) = H_{K_i}(M'))]\end{aligned}$$

For a secure hash function, $\mathbf{Adv}_{H}^{coll}(B)$ should be negligible (very small).

5.4 Proposed Scheme

5.4.1 Notations

In the following composition, we uses the following notations:

- $\|$, represents concatenation, for example $a\|b$ is a concatenation of a and b.
- $\oplus$, represents the binary XOR.
- $\lfloor\rfloor_\ell$, represents the last ℓ bits (bits on the right), for example $\lfloor\sigma_i\rfloor_\ell$ represents ℓ bits of σ_i from the right.
- ${}_\ell\lfloor\rfloor$, represents the first ℓ bits (bits on the left), for example ${}_\ell\lfloor\sigma_i\rfloor$ represents ℓ bits of σ_i from the right.

5.4.2 Primitives: VPSign, VPPla, VPVer

The basic form of our signature scheme is a variant of PSS with recovery originaly proposed by Bellare et al. [89]. The PSS with message recovery randomizes the signature while providing the possibility to reconstruct a large part of the signed message. We define a variant of PSS for signing and verification as VPSign and VPVer. We include another algorithm used for plausibility check we call VPPla.

As the original PSS, we need to assume, we have the hash functions as follows:

- $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{k_1}$,
- $g = g_1 \| g_2$, where
- $g_1 : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$, and
- $g_2 : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$.

VPSign, VPVer, and VPPla are defined in the following algorithms:

```
Algorithm $\mathsf{VPSign}_d(\sigma_{i-1}, m_i)$
  $s_{i-1} \leftarrow \lfloor \sigma_{i-1} \rfloor_\ell$
  $r_i \leftarrow \mathsf{H}(m_i)$
  $w_i \leftarrow \mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| r_i)$
  $r_i^* \leftarrow g_1(w_i) \oplus r_i$
  $\sigma_i^* \leftarrow g_2(w_i) \oplus_{n-\ell} \lfloor \sigma_{i-1} \rfloor$
  $\sigma_i \leftarrow (0 \| w_i \| r_i^* \| \sigma_i^*)^d \mod N$
  return $s_{i-1} \| \sigma_i$
```

```
Algorithm $\mathsf{VPPla}_e(\sigma_i)$
  $(b \| w_i \| r_i^* \| \sigma_i^*) = \sigma_i^e \mod N$
  $r_i = g_1(w_i) \oplus r_i^*$
  $\sigma'_{i-1} = g_2(w_i) \oplus \sigma_i^*$
  if $(w_i = \mathsf{H}(\sigma'_{i-1} \| r_i)$ and $b = 0)$ then
    return $\sigma'_{i-1}$
  else
    return REJECT
  end if
```

```
Algorithm $\mathsf{VPVer}_e(\sigma_i, m_i)$
  $(b \| w_i \| r_i^* \| \sigma_i^*) = \sigma_i^e \mod N$
  $r_i = g_1(w_i) \oplus r_i^*$
  $\sigma'_{i-1} = g_2(w_i) \oplus \sigma_i^*$
  if $(w_i = \mathsf{H}(\sigma'_{i-1} \| r_i)$ and $r = \mathsf{H}(m_i)$ and $b = 0)$ then
    return $\sigma'_{i-1}$
  else
    return REJECT
  end if
```

5.4.3 Signing the Sequence

We propose the method to sign a sequence of digital documents using the following algorithms:

KeyGen. The key generation algorithm uses the standard RSA algorithm to generates the public and private key pair (e, N) and (d, N) with a large modulus.

Sign. The signature algorithm works for a sequence of message $M = \{m_1, m_2, m_3,$ ..., $m_n\}$ and produces the output $\sigma = s_1 \| s_2 \| \cdots \| s_{n-1} \| \sigma_n$. We define the signature algorithm as follows:

```
Algorithm $\mathsf{Sign}_d(\{m_1, m_2, m_3, ..., m_n\})$
  $\sigma_0 \leftarrow \{0\}^n$
  $i \leftarrow 1$
  while $(i \leq n)$ do
    $s_{i-1} \| \sigma_i \leftarrow \mathsf{VPSign}_d(\sigma_{i-1}, m_i)$
    $i \leftarrow i + 1$
  end while
  return $s_1 \| s_2 \| \cdots \| s_{n-1} \| \sigma_n$
```

Append. We use the following algorithm to include a new member m_n to the existing sequence $\{m_1, m_2, m_3, ..., m_{n-1}\}$ and update the current signature to a new signature for the new sequence. The new member will be the latest member of the sequence.

```
Algorithm Append$_d(s_1\|s_2\|\cdots\|s_{n-2}\|\sigma_{n-1}, m_n)$
  $s_{n-1}\|\sigma_n \leftarrow$ VPSign$_d(\sigma_{n-1}, m_n)$
  return $s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n$
```

IsPlausible. Checking whether the signature is plausible.

```
Algorithm IsPlausible$_e(s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n)$
  $i \leftarrow n$
  while $(i \geq 1)$ do
    if (VPPla$_e(\sigma_i) =$ REJECT) then
      return FALSE
    end if
    $\sigma_{i-1} \leftarrow$ VPPla$_e(\sigma_i)\|s_{i-1}$
    $i \leftarrow i - 1$
  end while
  return TRUE
```

IsMember. Verification algorithm for a member m_j.

```
Algorithm IsMember$_e(s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n, m_j)$
  $i \leftarrow n$
  while $(i \geq 1)$ do
    if ($i \neq j$ and VPPla$_e(\sigma_i) =$ REJECT) then
      return FALSE
    else if ($i = j$ and VPVer$_e(\sigma_i, m_i) =$ REJECT) then
      return FALSE
    end if
    $\sigma_{i-1} \leftarrow$ VPPla$_e(\sigma_i)\|s_{i-1}$
    $i \leftarrow i - 1$
  end while
  return TRUE
```

IsLater. Checking whether m_k is included after m_j.

```
Algorithm IsLater$_e(s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n, m_j, m_k)$
  $i \leftarrow n$
  while $(i \geq 1)$ do
    if $((i \neq j$ and $i \neq k)$ and VPPla$_e(\sigma_i) =$ REJECT) then
      return REJECT
    else if $((i = j$ or $i = k))$ and VPVer$_e(\sigma_i, m_i) =$ REJECT) then
      return REJECT
    end if
    $\sigma_{i-1} \leftarrow$ VPPla$_e(\sigma_i)\|s_{i-1}$
    $i \leftarrow i - 1$
  end while
  if $(j < k)$ then
    return TRUE
  else
    return FALSE
  end if
```

Verify. Verifying whether the signature $s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n$ is a correct signature on $\{m_1, m_2, m_3, ..., m_n\}$.

```
Algorithm Verify$_e(s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n, \{m_1, m_2, m_3, ..., m_n\})$
  $i \leftarrow n$
  while $(i \geq 1)$ do
    if (VPVer$_e(\sigma_i, m_i) =$ REJECT) then
      return FALSE
    end if
    $\sigma_{i-1} \leftarrow$ VPVer$_e(\sigma_i, m_i)\|s_{i-1}$
    $i \leftarrow i - 1$
  end while
  return TRUE
```

5.4.4 Correctness of the signature scheme

For the n messages $\{m_1, ..., m_n\}$, we need to show that a correct signature $s_1\|s_2\|\cdots\|s_{n-1}\|\sigma_n$ will return **TRUE** during verification process, otherwise the verification algorithm will return **REJECT**.

At first, we need to show that $\mathsf{VPVer}_e(\sigma_i, m_i)$ will return **REJECT** if $s_{i-1}\|\sigma_i \neq \mathsf{VPSign}_d(\sigma_{i-1}, m_i)$. From the VPSign algorithm it is easy to check that because $\sigma_i \leftarrow (0\|w_i\|r_i^*\|\sigma_i^*)^d \mod N$, then σ_i is produced by a unique $(0\|w_i\|r_i^*\|\sigma_i^*)$, so that from the following equations

$$\begin{aligned} r_i &= g_1(w_i) \oplus r_i^* \\ {}_{n-\ell}\lfloor \sigma_{i-1} \rfloor &= g_2(w_i) \oplus \sigma_i^* \\ w_i &= \mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| r_i) \\ r_i &= \mathsf{H}(m_i) \end{aligned}$$

the value of ${}_{n-\ell}\lfloor \sigma_{i-1}$ and $r_i = \mathsf{H}(m_i)$ should be also unique because the values of $g_1(w_i)$ and $g_2(w_i)$ are fixed. The only possibility of the same signature produced by a different message is if we can find collision of H, so that we know m_i' where $\mathsf{H}(m_i') = \mathsf{H}(m_i)$. By assuming that the problem of finding m_i' is difficult, we can safely assume that the provided message message is indeed the original signed message. For the valid values of σ_i and m_i, $\mathsf{VPVer}_e(\sigma_i, m_i)$ will return a unique σ_{i-1}' (by definition, it means a valid signature).

Now, we need to show that if σ_i is correct, for m_{i-1}, we need to have a correct s_{i-1}, otherwise the verification function will return **REJECT**. By contradiction, assume that s_{i-1} is incorrect (different from the original one), because $\sigma_{i-1} = \sigma_{i-1}'\|s_{i-1}$, then $\left(0\|w_{i-1}\|r_{i-1}^*\|\sigma_{i-1}^*\right)$ will be different from the original one. To be accepted as the signature, we should check the following equations:

$$\begin{aligned} r_{i-1} &= g_1(w_{i-1}) \oplus r_{i-1}^* \\ {}_{n-\ell}\lfloor \sigma_{i-2} \rfloor &= g_2(w_{i-1}) \oplus \sigma_{i-1}^* \\ w_{i-1} &= \mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-2} \rfloor \| r_{i-1}) \\ r_{i-1} &= \mathsf{H}(m_{i-1}) \end{aligned}$$

We can continue the argument for the signature m_{i-2}, so that for $\{m_1, ..., m_n\}$, the signature $s_1\|s_2\| \cdots \|s_{n-1}\|\sigma_n$ should be correct.

5.4.5 Proving the order of the sequence

To prove the order of the sequence, we need to call the function **IsLater** which checks whether one of the two members m_j and m_k are included before or after the other. The algorithm **IsLater** verifies the order by first checking whether both members are included in the sequence and then checking which one is included before or after another. **IsLater** verifies the order by reversing the creation of the signature, so that we can get the original order of the messages. As proved in Section 5.5.2, a specific form of the signature can only represents a unique order of the message, so that **IsLater** will correctly verifies the order of the sequence.

5.4.6 Signature size

As described in Section 6.5, the signature on $\{m_1, m_2, m_3, ..., m_n\}$ is represented by $s_1 \| s_2 \| \cdots \| s_{n-1} \| \sigma_n$ where the size of σ_n is constant and the same as the size of the RSA signature while the size of $s_1 \| s_2 \| \cdots \| s_{n-1}$ is $(n-1) \times$(the size of s_i). For a setup as in the RSA-PSS with message recovery [89] where the size of $s \approx \frac{1}{4}$ of the standard RSA, we get the following: $\frac{(n-1)S}{4} + S = \frac{(n+2)S}{4}$ where S is the size of the standard RSA signature.

5.5 Security Proofs

We need to analyze and show the security of the signature scheme under *EEUF-CMA* and *OUF-CMA*.

5.5.1 Security under EEUF-CMA

The main difference of *EEUF-CMA* with the standard *EUF-CMA*, is that in the *EEUF-CMA* the attacker can forge the signature by showing only one of the signed messages to the verifier. The possible form of the signature on the sequence of message $M = \{m_1, ..., m_n\}$, is $s_1 \| s_2 \| \cdots \| s_{n-1} \| \sigma_n$. We need to show that by showing only one message $m_i \in M$ and the signature $s_1 \| s_2 \| \cdots \| s_{n-1} \| \sigma_n$, if the attacker can pass the IsPlausible and IsMember checks, then we can use the outputs of the attacker to solve the RSA problem.

The security proof is similar to the security proof of RSA-PSS [89], except that, the random r is substituted by the hash.

Theorem 5.3. *If the RSA problem is hard, then the proposed scheme as described in Section 6.5 is secure in the extended existential unforgeability under chosen message attack (EEUF-CMA) model.*

Proof. Let A be a forger who break the signature in EEUF-CMA, we show a simulator B which break the RSA.

We need to prove that if the attacker A can forge the signature in *EEUF-CMA*, we show that there exists a simulator B that can simulate A's environment without detected by A and then by using A's outputs, the simulator B can solve the RSA problem. This will contradict the fact that the RSA problem is hard.

In the following, we describe how to simulate A's environment which is indistinguishable from the real A's environment. Because the attacker A is given access to the random oracle H, g_1, and g_2 and the signing function $Sign$, simulator B should be able to simulate both of the random oracle and the signing function without having better information than A. A should produce a valid m_i* and j_i* and σ_i* and we need to show how B uses A's outputs to solve an RSA problem.

Without loss of generality, we assume that the attacker will forge the message m_n from the sequence $M = \{m_1, ..., m_n\}$, and $\{m_1, ..., m_{n-1}\}$ are inaccessible during verification. We should note that the other possible forms of forgery (forgery for m_i where $i \neq n$) can be easily derived from this form.

In this form of forgery, the simulator B only needs to simulate the hash output for m_n, for the hash of $\{m_1, ..., m_{n-1}\}$, we may assume that the forger get the outputs from other random oracles. In the worst case scenario (as used in this proof), the attacker sets the value of the hash for $\{m_1, ..., m_{n-1}\}$ directly. This is possible because during verification, the messages are inaccessible, so there is no way to check the correctness of the hashes.

Setup. The simulator B is given the signer public key (e, N) and σ_{i-1}, but B has no access to the private key d. The simulator B is also provided with a and later B should solve the RSA problem by computing $a^d \mod N$ from A's outputs. The simulator B starts by giving the forger algorithm A the public key (e, N) and the last signature σ_{i-1} (which is the signature of previous version of the document).

Answering Signature Queries.

During the signature queries $(s_{i-1} \| \sigma_i \leftarrow \mathsf{VPSign}_d(\sigma_{i-1}, m_i))$, the simulator B receives inputs σ_{i-1}, m_i. To answer the signature queries B sets a random signature x_i and sets $y_i = x_i^e \mod N$ where y_i has first bit 0. Then B sets $y_i = (0 \| w_i \| r_i^* \| \sigma_i^*)$, The simulator then derives the result of the hashes functions $H(m_i), w_i, g_1(w_i)$, and $g_2(w_1)$ as follows:

1. Set $\mathsf{H}(m_i) \xleftarrow{\$} \{0,1\}$, if $\mathsf{H}(m_j) = \mathsf{H}(m_i)$ for $j < i$ then **ABORT**.
2. $x_i \xleftarrow{\$} \{0,1\}$, $y_i \leftarrow x_i^e \mod N$, if the first bit of y_i is 0 then set $(0 \| w_i \| r_i^* \| \sigma_i^*) \leftarrow y_i$, otherwise repeat this step.
3. Set $\mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| \mathsf{H}(m_i)) = w_i$, if $w_j = w_i$ for $j < i$ then **ABORT**.
4. Set $g_1(w_i) = r_i^* \oplus \mathsf{H}(m_i)$, Set $g_2(w_i) = \sigma_i^* \oplus_{n-\ell} \lfloor \sigma_{i-1} \rfloor$, Set $g(w_i) = g_1(w_i) \| g_2(w_i)$,

The simulator should keep the pair of messages and the hashes consistently, so that the values can be retrieved later.

Answering H Queries.

1. the queries have either $\mathsf{H}(m_i)$ form, or $\mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| \mathsf{H}(m_i))$ form, without loss of generality we assume that no queries are repeated, so each query is unique. The simulator keeps all previous queries, so that it can detect if the queries is in the form $\mathsf{H}(m_i)$ or $\mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| \mathsf{H}(m_i))$, because σ_{1-1} is the result of previous sign queries, while $\mathsf{H}(m_i)$ is the result of previous H Queries.
2. If the query is the $\mathsf{H}(m_i)$-type one:
   (a) Set $\mathsf{H}(m_i) \xleftarrow{\$} \{0,1\}$, if $\mathsf{H}(m_j) = \mathsf{H}(m_i)$ for $j < i$ then **ABORT**. Otherwise return $\mathsf{H}(m_i)$
3. If the query is the $\mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| \mathsf{H}(m_i))$-type one:
   (a) $x_i \xleftarrow{\$} \{0,1\}$, $y_i \leftarrow a x_i^e \mod N$, if the first bit of y_i is 0 then set $(0 \| w_i \| r_i^* \| \sigma_i^*) \leftarrow y_i$, otherwise repeat this step.
   (b) Set $\mathsf{H}({}_{n-\ell}\lfloor \sigma_{i-1} \rfloor \| \mathsf{H}(m_i)) = w_i$, if $w_j = w_i$ for $j < i$ then **ABORT**.
   (c) Set $g_1(w_i) = r_i^* \oplus \mathsf{H}(m_i)$, Set $g_2(w_i) = \sigma_i^* \oplus_{n-\ell} \lfloor \sigma_{i-1} \rfloor$, Set $g(w_i) = g_1(w_i) \| g_2(w_i)$,

4. If the query is the $\mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|\mathsf{H}(m_i))$-type one:
   (a) $x_i \xleftarrow{\$} \{0,1\}$, $y_i \leftarrow ax_i^e \mod N$, if the first bit of y_i is 0 then set $(0\|w_i\|r_i^*\|\sigma_i^*) \leftarrow y_i$, otherwise repeat this step.
   (b) Set $\mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|\mathsf{H}(m_i)) = w_i$, if $w_j = w_i$ for $j < i$ then **ABORT**.
   (c) Set $g_1(w_i) = r_i^* \oplus \mathsf{H}(m_i)$, Set $g_2(w_i) = \sigma_i^* \oplus_{n-\ell} \lfloor\sigma_{i-1}\rfloor$, Set $g(w_i) = g_1(w_i)\|g_2(w_i)$,

After answering the queries, the simulator should keep the pair of messages and the hashes consistently, so that the values can be retrieved later.

Answering g Queries.

Answering $g(w_i)$ queries is simple, if $w_j = w_i$ for $j < i$, simply reply with the previous result $(g(w_j)$. Otherwise return random $g(w_i) \xleftarrow{\$} \{0,1\}$.

Output.

The forger ouputs a signature message pair $(\sigma_i^\star, m_i^\star)$ that has not been queried before, then the forger should have queried $\mathsf{H}(m_i)$ and $\mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|\mathsf{H}(m_i))$. If the signature is valid, the simulator computes $y_i^\star \leftarrow (\sigma_i^\star)^e \mod N$ and sets $y_i^\star$ as $(0\|w_i\|r_i^*\|\sigma_i^*)$.

The simulator checks the previous queries in the form $\mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|\mathsf{H}(m_i))$ and $\mathsf{H}(m_i)$, where $\mathsf{H}(m_i) = r^* \oplus g_1(w_i)$ and ${}_{n-\ell}\lfloor\sigma_{i-1}\rfloor = \sigma_i^* \oplus g_2(w_i)$. For this condition, because $y_i^\star = (0\|w_i\|r_i^*\|\sigma_i^*) = ax_i^e \mod N$, then the simulator outputs $\sigma_i^\star/x_i \mod N$.

Because $(y_i^\star)^d = ((\sigma_i^\star)^e)^d = (ax_i^e)^d \mod N$, then $\sigma_i^\star = x_i a^d \mod N$, so that $a^d \mod N = \sigma_i^\star/x_i \mod N$.

From the above argument, we show that if there exists a forger A who breaks our signature scheme then there exists a simulator B that can solve the RSA problem. □

5.5.2 Security under OUF-CMA

Theorem 5.4. *If the RSA problem is hard, then the proposed scheme as described in Section 6.5 is secure in the order unforgeability under chosen message attack (OUF-CMA) model.*

Proof. Assuming the scheme is secure under EEUF-CMA (the attacker cannot forge the unsigned documents), the attacker can only forge the order by re-arranging the signed messages and the valid signatures. To re-arrange the order of a signed sequence, the attacker should find the collision in the signature. That is, the attacker should find the condition $s_{i-1} \| \sigma_i = s_{j-1} \| \sigma_j$ where $i \neq j$ and $s_{i-1} \| \sigma_i \leftarrow \mathsf{VPSign}_d(\sigma_{i-1}, m_i), s_{j-1} \| \sigma_j \leftarrow \mathsf{VPSign}_d(\sigma_{j-1}, m_j)$.

We need to prove that the probability to find the collision can be reduced to the probability to find the collision of the hash function used in the scheme. First, we define the possibility to find the collision in the signature as follows:

$$\begin{aligned} \mathbf{Adv}^{ouf}_{Sign}(A) &= \Pr[d_i \xleftarrow{\$} D; (M, M') \xleftarrow{\$} A(d_i) : \\ &\qquad (Sign_{d_i}(M) = Sign_{d_i}(M'))] \end{aligned}$$

Then, we compute the probability of $\mathbf{Adv}^{ouf}_{Sign}(A)$ by using game-based proof (**G0** to **G4**) as follows:

G0. This game represents the original game. The attacker can query the valid pair of message-signature pair, in the end of the game, the attacker outputs $m_i, m_j, s_{i-1} \| \sigma_i = s_{j-1} \| \sigma_j$ where $i \neq j$.

```
$\sigma_0 \leftarrow \{0\}^n, d \xleftarrow{\$} D$
**for** $i \leftarrow 1...q$ **do**
    $s_{i-1} \leftarrow \lfloor \sigma_{i-1} \rfloor_\ell$
    $r \leftarrow \mathsf{H}(m_i)$
    $w_i \leftarrow \mathsf{H}(_{n-\ell} \lfloor \sigma_{i-1} \rfloor \| r)$
    $r_i^* \leftarrow g_1(w_i) \oplus r$
    $\sigma_i^* \leftarrow g_2(w_i) \oplus_{n-\ell} \lfloor \sigma_{i-1} \rfloor$
    $\sigma_i \leftarrow \left(0 \| w_i \| r_i^* \| \sigma_i^*\right)^d \mod N$
    **return** $s_{i-1} \| \sigma_i$
**end for**
**return** $m_i, m_j, s_{i-1} \| \sigma_i = s_{j-1} \| \sigma_j$ where $i \neq j$ and $m_i \neq m_j$
```

$$\mathbf{Adv}^{ouf}_{Sign}(A) \quad = \quad \Pr[G_0^A \Rightarrow 1]$$

G1. In this game, because the RSA is permutation, to find the collision, the attacker should find $(0\|w_i\|r_i^*\|\sigma_i^*)$, and $\big(0\|w_j\|r_j^*\|\sigma_j^*\big)$ where $(0\|w_i\|r_i^*\|\sigma_i^*) = \big(0\|w_j\|r_j^*\|\sigma_j^*\big)$.

```
$\sigma_0 \leftarrow \{0\}^n, d \xleftarrow{\$} D$
for $i \leftarrow 1...q$ do
    $s_{i-1} \leftarrow \lfloor\sigma_{i-1}\rfloor_\ell$
    $r \leftarrow \mathsf{H}(m_i)$
    $w_i \leftarrow \mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|r)$
    $r_i^* \leftarrow g_1(w_i) \oplus r$
    $\sigma_i^* \leftarrow g_2(w_i) \oplus_{n-\ell} \lfloor\sigma_{i-1}\rfloor$
    return $\leftarrow (0\|w_i\|r_i^*\|\sigma_i^*)$
end for
return $(0\|w_i\|r_i^*\|\sigma_i^*) = \big(0\|w_j\|r_j^*\|\sigma_j^*\big)$ where $i \neq j$
```

$$\begin{aligned}\Pr[G_0^A \Rightarrow 1] &= \Pr[G_1^A \Rightarrow 1]\\ \mathbf{Adv}_{Sign}^{ouf}(A) &= \Pr[G_1^A \Rightarrow 1]\end{aligned}$$

G2. In this game, we assume $H(m_i)$ produces distinct results r_i where the advantage of the adversary B finds a collision is $\mathbf{Adv}_H^{coll}(B)$.

```
$\sigma_0 \leftarrow \{0\}^n, d \xleftarrow{\$} D$
for $i \leftarrow 1...q$ do
    $s_{i-1} \leftarrow \lfloor\sigma_{i-1}\rfloor_\ell$
    $r \leftarrow r_i$
    $w_i \leftarrow \mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|r)$
    $r_i^* \leftarrow g_1(w_i) \oplus r$
    $\sigma_i^* \leftarrow g_2(w_i) \oplus_{n-\ell} \lfloor\sigma_{i-1}\rfloor$
    return $\leftarrow (0\|w_i\|r_i^*\|\sigma_i^*)$
end for
return $(0\|w_i\|r_i^*\|\sigma_i^*) = \big(0\|w_j\|r_j^*\|\sigma_j^*\big)$ where $i \neq j$
```

So, that

$$\begin{aligned}\Pr[G_1^A \Rightarrow 1] &= \Pr[G_2^A \Rightarrow 1] + \mathbf{Adv}_H^{coll}(B)\\ \mathbf{Adv}_{Sign}^{ouf}(A) &\leq \Pr[G_2^A \Rightarrow 1] + \mathbf{Adv}_H^{coll}(B)\end{aligned}$$

G3. In this game, again, we assume $\mathsf{H}({}_{n-\ell}\lfloor\sigma_{i-1}\rfloor\|r)$ produces distinct results t_i where the advantage of the adversary B finds a collision is $\mathbf{Adv}_H^{coll}(B)$.

```
$\sigma_0 \leftarrow \{0\}^n, d \xleftarrow{\$} D$
for $i \leftarrow 1...q$ do
    $s_{i-1} \leftarrow \lfloor \sigma_{i-1} \rfloor_\ell$
    $r \leftarrow r_i$
    $w_i \leftarrow t_i$
    $r_i^* \leftarrow g_1(w_i) \oplus r$
    $\sigma_i^* \leftarrow g_2(w_i) \oplus_{n-\ell} \lfloor \sigma_{i-1} \rfloor$
    return $\leftarrow (0\|w_i\|r_i^*\|\sigma_i^*)$
end for
return $(0\|w_i\|r_i^*\|\sigma_i^*) = \left(0\|w_j\|r_j^*\|\sigma_j^*\right)$ where $i \neq j$
```

So, that

$$\begin{aligned}
\Pr[G_2^A \Rightarrow 1] &\leq \Pr[G_3^A \Rightarrow 1] + \mathbf{Adv}_H^{coll}(B) \\
\mathbf{Adv}_{Sign}^{ouf}(A) &\leq (\Pr[G_3^A \Rightarrow 1] + \mathbf{Adv}_H^{coll}(B)) \\
&+ \mathbf{Adv}_H^{coll}(B) \\
\mathbf{Adv}_{Sign}^{ouf}(A) &\leq \Pr[G_3^A \Rightarrow 1] + 2 \cdot \mathbf{Adv}_H^{coll}(B)
\end{aligned}$$

G4. In this game, we simplify the notation, so we are only concerned about w_i, because in the **G4** we assume that w_i is defined as distinct t_i, we get the following game.

```
$\sigma_0 \leftarrow \{0\}^n, d \xleftarrow{\$} D$
for $i \leftarrow 1...q$ do
    $t_i \leftarrow$ distinct
    return $\leftarrow t_i$
end for
return $t_i = t_j$ where $i \neq j$
```

We get the following:

$$\begin{aligned}
\Pr[G_3^A \Rightarrow 1] &\leq \Pr[G_4^A \Rightarrow 1] \\
\Pr[G_4^A \Rightarrow 1] &= 0 \\
\Pr[G_3^A \Rightarrow 1] &= 0
\end{aligned}$$

In the end, we get the following result:

$$\begin{aligned}
\mathbf{Adv}^{ouf}_{Sign}(A) &\leq \Pr[G_3^A \Rightarrow 1] + 2 \cdot \mathbf{Adv}^{coll}_{H}(B) \\
\mathbf{Adv}^{ouf}_{Sign}(A) &\leq 0 + 2 \cdot \mathbf{Adv}^{coll}_{H}(B) \\
\mathbf{Adv}^{ouf}_{Sign}(A) &\leq 2 \cdot \mathbf{Adv}^{coll}_{H}(B)
\end{aligned}$$

□

5.6 Comparison of Our Scheme with the Other Schemes

In this section, we compare our scheme to the other possible schemes that can be used to sign the sequence of the documents (see Table 5.1). We compare the signature size, the possibility to check the order, whether the scheme allows verification without having access to all members, whether the scheme allows sequential updates and whether the schemes require a trusted party.

Table 5.1: Comparison with other signing methods

Scheme	Size of the Signature	Allow Checking Order	Allow Missing Members	Allow Sequential Update	Needs Trusted Party
RSA comb.	S	Yes	No	No	No
RSA ctr.	nS	Yes	Yes	Yes	Yes
S. Chain	nS	Yes	No	Yes	No
S. Aggr.	S	Yes	No	Yes	No
Our scheme	$\frac{(n+2)S}{k}$	Yes	Yes	Yes	No

We summarize the results as follows:

1. (RSA comb.) The first possible scheme is the plain RSA where we combine the messages and create one signature for the message. This method is efficient in term of the storage, however it has weakness because we cannot update sequentially and the verification needs access to all members of the sequence.

2. (RSA ctr.) We may use the plain RSA and include consecutive counters for each member. This scheme has a main advantage where we can verify each member without having access to any other members. However, this method

needs a large size of the signature, and needs a trusted party to guarantee the counters represents the original order.

3. (S. Chain.) The plain Signature chain method can also be used to sign the sequence, and its main advantage is we can generate the signature sequentially. However, we need to store a large size of the signature and also during verifiation we needs to have access to all members of the sequence.

4. (S. Aggr.) A promising solution is the signature aggregate where its main advantage is with a small size signature we can prove many members of the signature. However, its main problem is we need to access all members during verification of any member of the sequence.

5. (Our scheme.) Our scheme uses less storage than RSA with counter but needs a larger storage than the signature aggregate. The main advantages of our scheme are we allow sequential signing and also allow verification without having access to all members of the sequence. By assuming that our signature primitives (VPSign, VPPla, and VPVer) can recover $\frac{k-1}{k}$ the original message, the signature size is $\frac{(n+2)S}{k}$ where S is the size of a full RSA signature.

5.7 Conclusions

In this chapter, we have proposed a method to sign a sequence of digital documents by using a variant of signature with message recovery originally proposed by Bellare et al. [89] as the signature primitive. We defined the security model for the signature scheme, and showed that the scheme is secure in the model. The advantages of our method are:

1. Our signature scheme can verify the authenticity and also the order of each member of the documents.

2. During the verification of a signature, the scheme does not require access to all members of the documents.

3. The signature size is smaller than signing each member of the sequence using the standard RSA signature.

Chapter 6

Proxy Re-encryption for Symmetric Key Cryptography

6.1 Introduction

Cryptography is about the art and science to conceal the meaning of a text. A main technique in cryptography is encryption, which is used to transform the text (normally called the plaintext) into an unintelligent form (the ciphertext). The ciphertext can be securely transmitted in an insecure channel because an attacker who intercepts the ciphertext cannot infer anything about the text without having a secret decryption key. The encryption technique is also useful to secure data at rest, for example to protect the data at the local database or at an online database (the cloud). Any attacker who successfully bypass the security of the local or the online database cannot access the text without having access the decryption key.

Encryption schemes can be classified based on the keys used for encryption and decryption. Symmetric encryptions use exactly the same key for encryption and decryption while asymmetric encryptions use a pair of public and private keys. Symmetric encryptions are generally faster than asymmetric ones but we need to assume the decryption keys are securely transferred to the recipients.

Many symmetric encryption schemes have been proposed and most of the schemes are getting better in hiding the relationships between the plaintext, the key, and also the ciphertext. The existence of a function that can meaningfully alter the ciphertext (i.e., malleability) is generally regarded as a bad characteristics of a secure cryptosystem. However, in some situations we need to have a symmetric encryption scheme whose ciphertext can be efficiently transformed in a meaningful way. We show the applications of those symmetric encryption schemes in the local and online encrypted database as follows.

1. **Fast key update in an encrypted local database**. A user encrypts the local files or databases for better security measures. The problem is when the key is unintentionally leaked, the user needs to change the encryption key. With the current fast hardware, the user may use a simple method: decrypt-and-then-encrypt approach, by first decrypting the data with the old (leaked) key and then encrypting with a new key. However, if we need to re-encrypt a large size of data, it is desirable to have a faster method than executing these two costly computations (decrypt and encrypt).

2. **Key update in an encrypted outsourced database**. Recently, the cloud model (i.e., data as service model) has been implemented by many Internet companies. The data owner can store the data in the cloud and access the data from anywhere at anytime. However, without additional security mechanisms, an attacker (who breaks the security of the server) and also the company itself can do anything to the data: accessing, updating or removing the data. A promising solution is by encrypting the data before storing the data in the server, so that the access to the data is fully controlled by the data owner.

By encrypting the data, we can also implement encryption-based access control, where access to the data is granted by providing the decryption key. The problem is when the data owner needs to securely revoke an access, the data owner also needs to update the encryption key. Using a simple solution (decrypt-and-then-encrypt), re-encryption is a costly computation, because the data owner needs to download the data, decrypt the data and encrypt with the new key and then re-upload the data to the database. A promising solution is by securely delegating a re-encryption mechanism to a semi-trusted proxy in the server.

6.1.1 Security Model

We are concerned about an encryption scheme that can be used to encrypt the data as the other encryption modes (i.e., CBC, CTR). The typical usage is that the data owner encrypts the data and stores the data in a storage that can be accessed by some other users or malicious attackers. Our proposed scheme can be used to encrypt the data with some special features:

1. The encryption key can be updated directly, without having to decrypt the data with the previous key and encrypt with the new key.
2. The data owner can delegate the re-encryption process to a semi-trusted proxy (for example a cloud server) while the proxy is not allowed to access the decrypted form (plaintext). The semi-trusted proxy is a proxy which follows the algorithm correctly but we do not allow the proxy to access the sensitive data. An online database server may act as a proxy where the data owner may ask the proxy to update the key by sending the re-encryption key to the proxy.

In this usage model, we identify three types of attackers who has interest in attacking the encryption scheme.

1. **The outsiders**. In term of encryption security model, the outsiders are the normal attackers. The outsiders do not have access to any encryption keys. In the Chosen Plaintext Attack (CPA) model, we provide the outsiders an encryption oracle that can be used to collect the pair of chosen plaintexts/-ciphertexts before playing the indistinguishability game where the attackers should distinguish ciphertexts produced by some chosen plaintexts but not yet queried to the encryption oracle.
2. **The previous users**. In our model, the previous users are the most powerful attackers, because the previous users have access to at least one encryption key before re-encryption. In the re-encryption process, the data (plaintext) are not updated, so that we also need to assume that the previous users may have access to all existing plaintexts (by storing the decrypted data from the previous accesses). The only restriction to the previous users is that they do not have access to the current encryption and the re-encryption keys.
3. **The proxy**. The main feature of the encryption scheme is that we can delegate the re-encryption process (key update) to a semi-trusted proxy, but the proxy is not allowed to access to the encryption key. To re-encrypt the

data, the proxy should have access to the re-encryption keys. So, the proxy is more powerful than the outsiders (because it has more information – the re-encryption keys), but it is less powerful than the previous users.

6.1.2 Contributions

Our main contribution is: we propose a secure symmetric encryption scheme that supports fast key update and proxy re-encryption. The idea of our scheme is by first transforming the plaintext using an All or Nothing Transform (AONT) and exploits some of its characteristics to implement efficient and secure proxy re-encryptions scheme[1]. We prove the security of the scheme under the chosen plaintext attack security model. We also show that the scheme is more efficient than decrypt-and-then-encrypt method.

6.2 Related Work

6.2.1 Ciphertext Transformation and Proxy Re-encryption

Many researchers have proposed the method for direct transformation and proxy re-encryption in the asymmetric key encryption setting [6–10]. Blaze et al. [6] proposes a bidirectional proxy re-encryption based on ElGamal's encryption in their seminal paper. The scheme works with a generator g of prime order q, the private key $sk_a = a$ is a randomly selected element of $\mathbb{Z}_q^*$ and the public key pk_a is g^a. The ciphertext c_a for the message m is (mg^r, g^{ar}) that can be decrypted back using the private key a to get the plaintext $m = \frac{mg^r}{(g^{ar})^{1/a}}$. A proxy can directly converts c_a to ciphertext c_b (for the private key $sk_b = b \in \mathbb{Z}_q^*$ and public key $pk_b = g^b$) by using a re-encryption key $rk_{a \to b} = b/a$. This re-encryption is possible because $c_b = (mg^r, (g^{ar})^{b/a})$. This scheme is bidirectional, because the proxy can also compute $rk_{b \to a} = a/b$ and if the proxy colludes with one of the users, they can recover the private key of the other user (i.e., $(a/b) * b = a$).

[1] We can also implement more efficient proxy re-encryption using asymmetric ciphers by only encrypting a small part of AONT transforms. However, this scheme has a weakness because the previous users (who have access to the previous versions) may store this small encrypted part and use it for decrypting the message even if the encryption key in the actual data has been changed.

Some researchers [7–10] have improved the Blaze et al.'s scheme. An example is the work of Atenise et al. [7] that uses pairing. The scheme works with bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with security parameter the random generator $g \in \mathbb{G}_1$ and $Z = e(g, g) \in \mathbb{G}_2$. The public key of the first user is $pk_a = (Z^{a_1}, g^{a_2})$ and the private key is $sk_a = (a_1, a_2)$. The encrypted form of a message $m \in \mathbb{G}_2$ is $c_a = (g^r, mZ^{ra_1})$ for random $r \in Z_q$. The proxy can convert c_a to another ciphertext using the pairing property $(e(g_1^a, g_2^b) = e(g_1, g_2)^{ab})$ by computing $c_b = (e(g^r, g^{a_1 b}), mZ^{ra_1}) = (Z^{br'}, mZ^{r'})$, for $r' = a_1 r$ so that it can be decrypted by a second user with public key $pk_b = g^b$ and the private key is $sk_b = b$ to get $m = \frac{(mZ^{r'})}{(Z^{br'})^{1/b}}$.

The problem with these schemes is they cannot be applied directly to the local or online databases because, for performance reason, the database is normally encrypted using symmetric key encryption. Cook et al. suggest a solution for proxy re-encryption for symmetric ciphers by using double encryption [141]. In this scheme, the owner encrypts the data m with a key k_1 and re-encrypt with another key k_2 to get, for example the ciphertext $c = E_{k_2}(E_{k_1}(m))$. The re-encryption is done by the proxy for the second encryption by providing k_2 to the proxy. During re-encryption process, the proxy can not access the data m, because the data is protected by the first encryption layer. Using this method, the proxy needs to execute two costly computations: decrypting with the old key and encrypting with the new key.

Cook et al. [141] also showed that the symmetric ciphers that are closed under functional composition [142], where for encryption E and message m, there exist encryption keys k_1, k_2, k_3 so that $E_{k_3}(m) = E_{k_2}(E_{k_1}(m))$, has a security weakness because it is vulnerable to known plaintext attack only requiring $2^{|k|/2}$ rather than $2^{|k|}$ for brute force key search.

Another related work is the conversion method for Galois Counter Mode (GCM) mode [143]. It is not intended for proxy re-encryption but rather for fast re-encryption. In this method, the re-encryption needs to execute two encryptions, so the performance of this mechanism is not much better than decrypt-and-then-encrypt approach.

6.2.2 All or Nothing Transform

All or nothing transform (AONT) converts n blocks message $M = m[1], ..., m[n]$ into a pseudo message $M' = m[1]', ..., m[s]'$ for $s > n$ so that the original message cannot be recovered if any block of the pseudo message is missing. Rivest [144] proposed an AONT that converts the message by encrypting each block with a random key and xor-ing the random key with the hash of all blocks, so that the encryption key cannot be recovered without having all parts of the pseudo messages. Let F be a block cipher, K_0 is a fixed (and publicly known) key. The Rivest' scheme is defined in the algorithm $\mathcal{E}$-AONT and $\mathcal{D}$-AONT as follows.

```
Algorithm $\mathcal{E}$-AONT$^{F,F^{-1}}(m[1], \ldots, m[n])$
  $K' \xleftarrow{\$} \{0,1\}^l$
  for $i = 1$ to $n$ do
    $m'[i] \leftarrow m[i] \oplus F_{K'}(i)$
    $h[i] \leftarrow F_{K_0}(m'[i] \oplus i)$
  end for
  $m'[n+1] = K' \oplus h[1] \oplus h[2] \oplus \cdots \oplus h[n]$
  return $m'[1], \ldots, m'[n+1]$
```

```
Algorithm $\mathcal{D}$-AONT$^{F,F^{-1}}(m'[1], \ldots, m'[n+1])$
  for $i = 1$ to $n$ do
    $h[i] \leftarrow F_{K_0}(m'[i] \oplus i)$
  end for
  $K' = m'[n+1] \oplus h[1] \oplus h[2] \oplus \cdots \oplus h[n]$
  for $i = 1$ to $n$ do
    $m[i] \leftarrow m['i] \oplus F_{K'}(i)$
  end for
  return $m[1], \ldots, m[n]$
```

The Rivest scheme is a specific form of the Optimal Asymmetric Encryption Padding (OAEP) proposed by Bellare et al.[145]. OAEP is intended as a padding method before encrypting with RSA. Let $G : \{0,1\}^{k_0} \rightarrow \{0,1\}^n$ be a random generator, and $H : \{0,1\}^n \rightarrow \{0,1\}^{k_0}$ is a hash function, and then for the message m and the parameter n and k, OAEP is defined in the following algorithm:

```
Algorithm $\mathcal{OAEP}^{G,H}(m, k_0, n)$
    $r \xleftarrow{\$} \{0,1\}^{k_0}$
    $k_1 \leftarrow n - \mathsf{length}(m)$
    $x \leftarrow m||0^{k_1} \oplus G(r)$
    $y \leftarrow r \oplus H(x)$
    return $x||y$
```

Boyko proved that OAEP is secure in several forms of security definition including adaptive semantic security [146]. Because Rivest's AONT is a form of OAEP, it is conceivable that it is also secure in these models [146]. Canetti et al. proposed some All-or-Nothing Transform schemes [147] while Dodis el al. proposed exposure resilient cryptography [148]. Desai proposed CTRT, that is a construction of AONT similar to Rivest's scheme based on CTR mode of encryption [149]. The main advantage is that the scheme needs only one ”pass” rather than two ”passes” as in the Rivest's scheme. Stinson provided an AONT scheme in the context of unconditional security [150]. More recently, Boneh et al. proposed key homomorphic Pseudorandom Function (PRF) that is secure in the standard model. The main construction is based on the learning with errors (LWE) problem [151].

6.2.3 Our Original Scheme

The scheme proposed in this chapter is our second attempt to develop a proxy re-encryption scheme in symmetric cryptography. The original version of the scheme described in this chapter has been published in [152]. In the original version, we used the plain All or Nothing Transform (AONT) by Rivest [144] and only showed the security proof in the Known Plaintext Attack Model while in this new version we use a modified/variant of All or Nothing Transform (AONT) (as described in Section 6.4.1) and proved the scheme is secure under the Chosen Plaintext Attack (CPA).

The other improvements to the original scheme are the usage of CTR-like encryption mode and a security proof of the key generation algorithm. All of the security proofs in this latest version are also treated more formally.

6.3 Preliminaries

6.3.1 Notion of Security

Security of a symmetric encryption scheme can be defined in term of Left-or-Right Indistinguishability [75] in which the attacker should decide whether he/she accesses a left (encrypting the left part of the message pair M_0, M_1) or a right (encrypting the right part of the message pair M_0, M_1) oracle as follows.

Definition 6.1 (LOR-CPA [75])**.** Let $\mathcal{SE} = \mathcal{K}, \mathcal{E}, \mathcal{D}$ be a symmetric encryption scheme. Let $b \in \{0, 1\}$ and $k \in \mathsf{N}$. Let A_{cpa} be an adversary that has access to the oracle $\mathcal{E}_K(\mathcal{LR}(\cdot\,, \cdot, b))$ that takes inputs (x_0, x_1) and produces the output $C \leftarrow \mathcal{E}_K(x_b)$. We consider the following experiment:

```
Experiment $\mathbf{Exp}^{lor-cpa-b}_{\mathcal{SE},A_{cpa}}(k)$:
    $K \overset{R}{\leftarrow} \mathcal{K}(k)$
    $d \leftarrow A_{cpa}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))}(k)$
    return $d$
```

We define the advantage of the adversaries as:

$$\begin{aligned}\mathbf{Adv}^{lor-cpa-b}_{\mathcal{SE}}(A) &= \Pr\left[\mathbf{Exp}^{lor-cpa-1}_{\mathcal{SE},A_{cpa}}(k) = 1\right] \\ &- \Pr\left[\mathbf{Exp}^{lor-cpa-0}_{\mathcal{SE},A_{cpa}}(k) = 1\right]\end{aligned}$$

The following conversion is useful to calculate the probability of LOR-CPA. Let $\Pr[Guess \Rightarrow true]$ be the probability that the adversary correctly guesses the left or right oracle, then:

$$\mathbf{Adv}^{lor-cpa-b}_{\mathcal{SE}}(A) = 2 \cdot \Pr[Guess \Rightarrow true] - 1$$

Proof.

$$
\begin{aligned}
&\Pr[Guess \Rightarrow true] \\
&= \Pr[Guess \Rightarrow 1|b=1]\cdot\frac{1}{2} + \Pr[Guess \Rightarrow 0|b=0]\cdot\frac{1}{2} \\
&= \Pr[Guess \Rightarrow 1|b=1]\cdot\frac{1}{2} + \left(1 - \Pr[Guess \Rightarrow 1|b=0]\cdot\frac{1}{2}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cdot(\Pr[Guess \Rightarrow 1|b=1] - \Pr[Guess \Rightarrow 1|b=0]) \\
&= \frac{1}{2} + \frac{1}{2}\cdot\left(\mathbf{Adv}_{\mathcal{SE}}^{lor-cpa-b}(A)\right)
\end{aligned}
$$

The security of an All or Nothing Transform (AONT) has been defined in [149]. In this model, the adversary can choose l blocks of message M and sends the blocks to an oracle that replies with the corresponding pseudomessage of M but one block of the pseudomessages is removed. Then, the oracle randomly chooses from two blocks where one of them is a part of the pseudomessage (which has been removed) and the other one is a random bits with the same length. The adversary needs to guess whether the block is a part of the pseudomessage or the random bits.

Definition 6.2 (AON-CPA [149])**.** Let $\mathcal{AO} = \mathcal{E}, \mathcal{D}$ be an AONT of block numbers l. Let $b \in \{0,1\}$ and A_{cpa} be an adversary that has access to the oracle $\mathcal{E}(\mathcal{AO}(\cdot))$ that takes inputs M and produces the pseudomessages $M' \leftarrow \mathcal{E}(M)$. We consider the following experiment:

```
Experiment $\mathbf{Exp}_{\mathcal{AO},A_{cpa}}^{aon-cpa}(b)$:
  $(M) \leftarrow A_{cpa}(\mathsf{find})$
  $y_0 \leftarrow \mathcal{E}(M)$
  $y_1 \overset{R}{\leftarrow} \{0,1\}^{|y_0|}$
  $d \leftarrow A_{cpa}^{\mathcal{Y}}(\mathsf{guess})$
  return $d$
```

Where $\mathcal{Y}$ takes a block number n and return $y_b[n]$ that is the n-th block of either y_0 or y_1 (chosen randomly).

We define the advantage of the adversaries as:

$$
\begin{aligned}
\mathbf{Adv}_{\mathcal{AO}}^{aon-cpa-b}(A) &= \Pr\left[\mathbf{Exp}_{\mathcal{AO},A_{cpa}}^{aon-cpa}(1) = 1\right] \\
&- \Pr\left[\mathbf{Exp}_{\mathcal{AO},A_{cpa}}^{aon-cpa}(0) = 1\right]
\end{aligned}
$$

With a similar argument to the LOR security definition, we can derive the advantage of the adversary to an AONT:

$$
\mathbf{Adv}_{\mathcal{AO}}^{aon-cpa-b}(A) = 2 \cdot \Pr\left[\mathbf{Exp}_{\mathcal{AO},A_{cpa}}^{aon-cpa}(b) = b\right] - 1
$$

6.3.2 PRF and PRP Advantages

Let $\mathcal{F}^{\ell,L}$ be a set of all functions from $\sum^{\ell}$ to $\sum^{L}$ and $\mathcal{P}^{\ell}$ be a set of all permutations on $\sum^{\ell}$, and PRF and PRP advantages are defined as follows:

Definition 6.3 (PRF Advantage)**.** Let F be a pseudorandom function family, that is a collections of functions $F_{K_i} : \sum^{\ell} \to \sum^{L}$, and the advantage of an adversary B to distinguish the outputs of F from a random function R is defined as follows:

$$
\begin{aligned}
\mathbf{Adv}_{F}^{prf}(B) &= \Pr[K_i \xleftarrow{\$} K : B^{F_{K_i}(\cdot)} = 1] \\
&- \Pr[R \xleftarrow{\$} \mathcal{F}^{\ell,L} : B^{R(\cdot)} = 1]
\end{aligned}
$$

Definition 6.4 (PRP Advantage)**.** Let E be a pseudorandom permutation family, that is a collections of functions $E_{K_i} : \sum^{\ell} \to \sum^{\ell}$, and the advantage of an adversary B to distinguish the outputs of E from a random permutation P is defined in the following formula:

$$
\begin{aligned}
\mathbf{Adv}_{E}^{prp}(B) &= \Pr[K_i \xleftarrow{\$} K : B^{E_{K_i}(\cdot)} = 1] \\
&- \Pr[P \xleftarrow{\$} \mathcal{P}^{\ell} : B^{P(\cdot)} = 1]
\end{aligned}
$$

6.3.3 Difference Lemma

In some of our security proofs, we use the game playing proof method as described in [99, 153]. The following is a lemma that is useful in the game playing proof method.

Lemma 6.5 (Difference Lemma [99, 100])**.** *Let A, B, F be events defined in some probability distribution, and if $A \wedge \neg F \iff B \wedge \neg F$, then $|\Pr[A] - \Pr[B]| \leq \Pr[F]$*

Proof.

$$\begin{aligned} |\Pr[A] - \Pr[B]| &= |\Pr[A \wedge F] + \Pr[A \wedge \neg F] \\ &\quad - \Pr[B \wedge F] - \Pr[B \wedge \neg F]| \\ &= |\Pr[A \wedge F] - \Pr[B \wedge F]| \leq \Pr[F] \end{aligned}$$

Bellare et al. also describe a similar lemma in [153] which is used in the game-playing proof method.

6.4 The Primitives

In this section we describe the primitives that are used as the building blocks for our encryption scheme.

6.4.1 All or Nothing Transform (AONT)

We propose a variant of Rivest's AONT as shown in the following algorithm ($\mathcal{E}$-AONTH and $\mathcal{D}$-AONTH). The main difference to the original Rivest scheme is: we include another "pass" so that the attacker cannot control the outputs of AONT even if he/she knows the key K'. We also use a hash function in the second "pass" rather than an encryption with a "fixed key". The differences are highlighted by boxes. Another difference is: we also include another variable *ctr*, that is used to generate unique counter for each block.

```
Algorithm $\mathcal{E}$-AONTH$^{F,F^{-1},H}(ctr, m[1], \ldots, m[n])$
    $K' \xleftarrow{\$} \{0,1\}^l$
    for $i = 1$ to $n$ do
        $x[i] \leftarrow m[i] \oplus F_{K'}(ctr + i)$
    end for
    $m'[n+1] \leftarrow K' \oplus \boxed{H(x[1] \cdots x[n])}$
    for $i = 1$ to $n$ do
        $\boxed{m'[i] \leftarrow x[i] \oplus H(m'[n+1] \oplus (ctr + i))}$
    end for
    return $ctr, m'[1], \ldots, m'[n+1]$
```

```
Algorithm $\mathcal{D}$-AONTH$^{F,F^{-1},H}(ctr, m'[1], ..., m'[n+1])$
    for $i = 1$ to $n$ do
        $\boxed{x[i] \leftarrow m'[i] \oplus H(m'[n+1] \oplus (ctr + i))}$
    end for
    $K' \leftarrow m'[n+1] \oplus \boxed{H(x[1] \cdots x[n])}$
    for $i = 1$ to $n$ do
        $m[i] \leftarrow x[i] \oplus F_{K'}(ctr + i)$
    end for
    return $m[1], \ldots, m[n]$
```

The security of $AONTH$ in shown in the following theorem.

Theorem 6.6 (Security of AONTH)**.** *Let H in $AONTH$ defined in Section 6.4.1 be a pseudorandom function, the advantages of an adversary A attacking $AONTH$ in AON-CPA security model as defined in Section 6.3 is at most:*

$$\mathsf{Adv}^{\mathsf{aon-cpa}}_{\mathcal{AO}}(A) \leq 2 \cdot \mathsf{Adv}^{prf}_{F}(B)$$

Proof. Basically, the argument is as follow. The AONTH outputs the pseudomessages blocks $m'[i]$ for $1 \leq i \leq n$

$$m'[i] = x[i] \oplus H(m'[n+1] \oplus (ctr + i))$$

Because $x[i] \leftarrow m[i] \oplus F_{K'}(ctr + i)$, then we can conclude that $x[i]$ and $m'[i]$ are random with the advantages of the attacker to distinguish $m'[i]$ from the random bits is $\mathsf{Adv}_F^{prf}(B)$. As of the $(n+1)$-block:

$$m'[n+1] = K' \oplus H(x[1] \cdots x[n])$$

Because K' is chosen at random, for any $H(x[1] \cdots x[n])$, we can conclude that $m'[n+1]$ should be indistinguishable for random. So, that if the missing block is $m'[i]$ for $1 \leq i \leq n$ the advantage of the attacker is $\mathsf{Adv}_F^{prf}(B)$, but if the missing block is $m'[n+1]$, the advantage of the attacker is 0. The complete proof is shown in Section 6.9.2. □

6.4.2 The functions $\mathcal{PE}, \mathcal{DP}$, and $\mathcal{FC}$

Two basic functions used in our scheme are permutation $\mathcal{PE}$ and its inverse $\mathcal{DP}$. A permutation $\mathcal{PE}$ is a bijection function that takes two input sequences of the same size n. The first sequence is the permutation key $p[1], ..., p[n]$ where $1 \leq p[i] \leq n$ for $1 \leq i \leq n$ and $p[a] \neq p[b]$ for $a \neq b$. The second sequence is any sequence $x[1], ..., x[n]$ where each element has the same size $l (\geq 1)$. The permutation $\mathcal{PE}$ transforms the second input sequence by changing the order of the sequence according to the first input sequence (permutation key). For example a permutation $\mathcal{PE}_{(4,2,3,1)}(a, b, c, d)$ transforms the the second input into (d, b, c, a). The $\mathcal{PE}$ algorithm is shown as follows.

```
Algorithm $\mathcal{PE}_{p[1],...,p[n]}(x[1], \ldots, x[n])$
  for $i = 1$ to $n$ do
    $x'[i] \leftarrow x[p[i]]$
  end for
  return  $x'[1], \ldots, x'[n]$
```

The inverse of permutation $\mathcal{DP}$ is a bijection function that takes two inputs as $\mathcal{PE}$. The difference is the $\mathcal{DP}$ converts back the second sequence that has been permuted using the $\mathcal{PE}$. An example is $\mathcal{DP}_{(4,2,3,1)}(d, b, c, a) = (a, b, c, d)$. The $\mathcal{DP}$ algorithm is shown as follows.

```
Algorithm $\mathcal{DP}_{p[1],\ldots,p[n]}(x[1],\ldots,x[n])$
  for $i = 1$ to $n$ do
    $x'[p[i]] = x[i]$
  end for
  return $x'[1],\ldots,x'[n]$
```

An output of a permutation can be converted directly to an output of another permutation by finding a conversion key. The conversion key p_C of a permutation key p_A to permutation key p_B is a permutation key so that for each input sequence x, $\mathcal{PE}_{p_B}(x) = \mathcal{PE}_{p_C}(\mathcal{PE}_{p_A}(x))$. We can find the conversion key p_C if we have p_A and p_B by using the following algorithm.

```
Algorithm $\mathcal{FC}(p_A[1],\ldots,p_A[n],p_B[1],\ldots,p_B[n])$
  for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
      if $p_A[i] = p_B[j]$ then
        $p_C[j] \leftarrow i$
        break
      end if
    end for
  end for
  return $p_C[1],\ldots,p_C[n]$
```

6.4.3 Permutation Key Generator (PGen)

The permutation key generator is used to generate permutation keys where each of them consists of a sequence of distinct numbers from 1 to n. We implement the function using a deterministic encryption function $F_K(p)$ that takes an input K and a plaintext p using the following algorithm. To generate a random sequence of numbers from 1 to n, first we generate two sequences, the first one is the sequence from 1 to n, the second one is a sequence produced by encrypting the first sequence with F_K. We change the order of the first sequence based on the order of the second sequence. The permutation key generator is implemented in the Algorithm $\mathcal{PG}$.

```
Algorithm $\mathcal{PG}_K(n)$
  for $i = 1$ to $n$ do
    $p[i] = i$
    $tmp[i] = F_K(i)$
  end for
  QuickSortPlus($p, tmp, 1, n$)
  return $p$

  Function QuickSortPlus($p, tmp, lo, hi$)
  if $lo < hi$ then
    $q \leftarrow$PartitionPlus($p, tmp, lo, hi$)
    QuickSortPlus($p, tmp, lo, q - 1$)
    QuickSortPlus($p, tmp, q + 1, hi$)
  end if
  EndFunction
  Function PartitionPlus($p, tmp, lo, hi$)
  $pivot \leftarrow tmp[hi]$
  $i \leftarrow lo$
  for $j = lo$ to $hi - 1$ do
    if $tmp[j] \leq pivot$ then
      swap $tmp[i]$ with $tmp[j]$
      swap $p[i]$ with $p[j]$
      $i \leftarrow i + 1$
    end if
  end for
  swap $tmp[i]$ with $tmp[hi]$
  swap $p[i]$ with $p[hi]$
  return $i$
  EndFunction
```

Security of $\mathcal{PG}_K(n)$. We need to argue that $\mathcal{PG}_K(n)$ is secure, that is for a random K, the algorithm $\mathcal{PG}_K(n)$ outputs a random sequence of numbers from 1 to n which is indistinguishable from a random permutation of the numbers from 1 to n. The argument is as follows: for a secure F and ℓ bits size of outputs, each output of $F_K(i)$ for $1 \leq i \leq n$ will be distributed randomly with the values from 0 to $2^\ell - 1$, so that if we sort the results based on the values of the outputs, we will get a permutation that is indistinguishable to the random permutation, because for each $F_K(a)$ and $F_K(b)$ where $a \neq b$, $1 \leq a \leq n$, $1 \leq b \leq n$, then $F_K(a) \neq F_K(b)$, and there is the same chance that either $a < b$ or $a > b$.

6.5 The Proposed Scheme

6.5.1 Definition

Let $m[1], m[2], ..., m[n]$ be a sequence of n blocks of message where the size of each $m[i]$ is ℓ bits. The encryption algorithm $\mathcal{PR}$ consists of six algorithms $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{E}$, $\mathcal{D}$, $\mathcal{RG}$, $\mathcal{RE}$ where:

- $\mathcal{G}_1$ is a key generation algorithm that produces random keys to be used by $\mathcal{G}_2$
- $\mathcal{G}_2$ is a key generation algorithm that produces random keys to be used by $\mathcal{E}$ and $\mathcal{D}$
- $\mathcal{E}$ is the encryption algorithm that converts n input blocks $m[1], m[2], ..., m[n]$ to s output ciphertext blocks $c[1], c[2], ..., c[s]$ for $s \geq n$
- $\mathcal{D}$ is the decryption algorithm that transforms the ciphertext $c[1], c[2], ..., c[s]$ back into the plaintext $m[1], m[2], ..., m[n]$
- $\mathcal{RG}$ is an algorithm to generate keys for re-encryption algorithm $\mathcal{RE}$
- $\mathcal{RE}$ is the re-encryption algorithm that transforms the ciphertext $c[1]^A$, $c[2]^A$,..., $c[s]^A$ encrypted with private key K_A into ciphertext $c[1]^B$, $c[2]^B$,...,$c[s]^B$ encrypted with private key K_B

6.5.2 The Scheme

The proxy encryption algorithm $\mathcal{PR} = (\mathcal{G}_1, \mathcal{G}_2, \mathcal{E}, \mathcal{D}, \mathcal{RG}, \mathcal{RE})$ works on $\ell \times n$ bits message $m[1], ..., m[n]$ where the message is divided into n blocks with size ℓ. Combination of the key generators $\mathcal{G}_1$ and $\mathcal{G}_2$ produce three random permutation keys P_1, P_2, P_3 that are later used by $\mathcal{E}$ and $\mathcal{D}$ for encryption and decryption. Encryption algorithm $\mathcal{E}$ works by first converting the plaintext into AONT's pseudomessage, and then uses three permutation keys P_1, P_2, P_3 to produce the ciphertext with a random initialization vector (iv). The decryption algorithm $\mathcal{D}$ is the inverse of the $\mathcal{E}$. The re-encryption key generator $\mathcal{RG}$ produces the re-encryption keys that are later used by the re-encryption function $\mathcal{RE}$ to update the encryption key. We show the detail of each algorithm as follows.

```
Algorithm $\mathcal{G}_1(\ell)$
  $K_1 \xleftarrow{\$} \{0,1\}^\ell$
  $K_2 \xleftarrow{\$} \{0,1\}^\ell$
  $K_3 \xleftarrow{\$} \{0,1\}^\ell$
  return $K_1, K_2, K_3$
```

```
Algorithm $\mathcal{G}_2(K_1, K_2, K_3, \ell, n)$
  $P_1 \leftarrow \mathcal{PG}_{K_1}(\ell)$
  $P_2 \leftarrow \mathcal{PG}_{K_2}(\ell)$
  $P_3 \leftarrow \mathcal{PG}_{K_3}(n)$
  return $P_1, P_2, P_3$
```

```
Algorithm $\mathcal{E}(K_1, K_2, K_3, ctr, m[1], ..., m[n], \ell, n)$
  $(P_1, P_2, P_3) \leftarrow \mathcal{G}_2(K_1, K_2, K_3, \ell, n)$
  $iv \xleftarrow{\$} \{0,1\}^\ell$
  $ctr, m'[1], ..., m'[n+1] \leftarrow \mathcal{E}\text{-AONTH}(ctr, m[1], ..., m[n])$
  $m''[1], ..., m''[n] \leftarrow \mathcal{PE}_{P_3}(m'[1], ..., m'[n])$
  $c[0] \leftarrow \mathcal{PE}_{P_1}(m'[n+1][1...\ell]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])$
  for $i = 1$ to $n$ do
    $c[i] \leftarrow (\mathcal{PE}_{P_1}(m''[i][1...\ell])$
        $\oplus \mathcal{PE}_{P_2}(c[i-1][1...\ell]))$
  end for
  return $ctr, iv, c[0]...c[n]$
```

```
Algorithm $\mathcal{D}(K_1, K_2, K_3, ctr, iv, c[0], ..., c[n], \ell, n)$
  $(P_1, P_2, P_3) \leftarrow \mathcal{G}_2(K_1, K_2, K_3, \ell, n)$
  for $i = n$ to $1$ do
    $m[i]'' \leftarrow \mathcal{DP}_{P_1}(c[i] \oplus \mathcal{PE}_{P_2}(c[i-1][1...\ell]))$
  end for
  $m'[n+1] = \mathcal{DP}_{P_1}(c[0][1...\ell] \oplus \mathcal{PE}_{P_2}(iv[1...\ell]))$
  $m'[1], ..., m'[n] \leftarrow \mathcal{DP}_{P_3}(m''[1], ..., m''[n])$
  $m[1], ..., m[n] \leftarrow \mathcal{D}\text{-AONTH}(ctr, m'[1], ..., m'[n+1])$
  return $m[1], ..., m[n]$
```

```
Algorithm $\mathcal{RG}(K_1, K_2, K_3, \ell, n)$
  $(P_1, P_2, P_3) \leftarrow \mathcal{G}_2(K_1, K_2, K_3, \ell, n)$
  $(K'_1, K'_2, K'_3) \leftarrow \mathcal{G}_1(\ell)$
  $(P'_1, P'_2, P'_3) \leftarrow \mathcal{G}_2(K'_1, K'_2, K'_3, \ell, n)$
  $CK_1 \leftarrow \mathcal{FC}(P_1, P'_1)$
  $CK_3 \leftarrow \mathcal{FC}(P_3, P'_3)$
  return $CK_1, K_2, K'_2, CK_3$
```

```
Algorithm $\mathcal{RE}(CK_1, K_2, K'_2, CK_3, ctr, iv, c[0], ..., c[n], \ell, n)$
  $P_2 \leftarrow \mathcal{PG}_{K_2}(\ell)$, $P'_2 \leftarrow \mathcal{PG}_{K'_2}(\ell)$
  for $i = n$ to 1 do
    $c'[i] \leftarrow \mathcal{PE}_{CK_1}(c[i] \oplus \mathcal{PE}_{P_2}(c[i-1][1...\ell]))$
  end for
  $iv' \xleftarrow{\$} \{0,1\}^\ell$
  $c''[1], ..., c''[n] \leftarrow \mathcal{PE}_{CK_3}(c'[1], ..., c'[n])$
  $c''[0] = \mathcal{PE}_{CK_1}(c[0] \oplus \mathcal{PE}_{P_2}(iv[1...\ell])))$
       $\oplus \mathcal{PE}_{P'_2}(iv'[1...\ell]))$
  $c[0] \leftarrow c[0]''$
  for $i = 1$ to $n$ do
    $c[i] \leftarrow c[i]'' \oplus \mathcal{PE}_{P'_2}(c''[i-1][1...\ell])$
  end for
  return $ctr, iv', c[0], ..., c[n]$
```

To encrypt a large message, the data owner generates the encryption keys by calling $\mathcal{G}_1$, and then divides the message into large blocks with size $\ell \times n$ each and assigns a unique counter *ctr* for each large block where the different of the *ctr* between consecutive large blocks is at least n. Each large block is then divided into n smaller ones with size ℓ bits for each block. Each large block is encrypted by executing the encryption algorithm $\mathcal{E}$ with inputs *ctr* and the blocks $m[1], \ldots, m[n]$. Figure 6.1 provides an illustration of the encryption algorithm $\mathcal{E}$.

At a later time, the data owner may need to update/change the encryption keys. First, he/she calls $\mathcal{G}_1$ to generate a new key, and call $\mathcal{RG}_1$ to generate the re-encryption key. If the encrypted data is stored in the local storage, the data owner may simply execute the re-encryption algorithm $\mathcal{RE}$ by him/herself to update encryption key. Alternatively, if the data is stored in an online storage, he/she may need to send the re-encryption key to the proxy. The proxy executes the

$$\begin{array}{cccc}
ctr & m[1] & \cdots & m[n] \\
 & & \downarrow \mathcal{E}\text{-AONTH} & \\
m'[n+1] & \{m'[1] & \cdots & m'[n]\} \\
 & & \downarrow \mathcal{PE}_{P_3} & \\
\downarrow & \searrow & \searrow\swarrow & \swarrow \\
m'[n+1] & \{m''[1] & \cdots & m''[n]\} \\
\downarrow \mathcal{PE}_{P_1} & \downarrow \mathcal{PE}_{P_1} & \downarrow \mathcal{PE}_{P_1} & \downarrow \mathcal{PE}_{P_1} \\
m'''[n+1] & m'''[1] & \cdots & m'''[n] \\
\downarrow & \downarrow & & \downarrow \\
iv \overset{\mathcal{PE}_{P_2}}{\rightarrow} \oplus = c[0] & \overset{\mathcal{PE}_{P_2}}{\rightarrow} \oplus = c[1] & \cdots \rightarrow & c[n-1] \overset{\mathcal{PE}_{P_2}}{\rightarrow} \oplus = c[n]
\end{array}$$

FIGURE 6.1: Illustration of the encryption of a large block ($\ell \times n$ bits)

re-encryption algorithm $\mathcal{RE}$ to update the key without the need to decrypt the data or accessing the previous encryption keys. The users who previously were granted access to the data by the data owner, cannot decrypt the new ciphertext without having the new encryption key. However he/she (the previous user) may keep some or all data that is accessible from previous accesses.

Encryption algorithm needs to call $\mathcal{E}$-AONTH function, and also permutation at the level of blocks and bits. For example $m''[1]...m''[n] \leftarrow \mathcal{PE}_{P_3}(m'[1]...m'[n])$ permutes many blocks $m'[1]...m'[n]$ at the level of blocks while $\mathcal{PE}_{P_1}(m''[i][1...\ell])$ permutes the block $m''[i]$ at the level of bits where the size of element which is permuted is one bit. Permutations at the level of blocks move a block (ℓ bits) to another position in a sequence of blocks, while permutations at the level of bits move only one bit to another position in one block.

6.5.3 Correctness of the Re-encryption Function $\mathcal{RE}$

It is easy to check that the decryption $\mathcal{D}$ is the inverse of the encryption function $\mathcal{E}$. In this section, we show that the re-encryption algorithm $\mathcal{RE}$ correctly converts a ciphertext encrypted with keys P_1, P_2, P_3 to another ciphertext encrypted with keys P_1', P_2', P_3'.

The ciphertext of each block before re-encryption is produced from AONT outputs $(m'[1]...m'[n+1])$ as follows:

$$\begin{aligned} m''[1] \quad ... \quad & m''[n+1] \leftarrow \mathcal{PE}_{P_3}(m'[1]...m'[n+1]) \\ c[0] \quad \leftarrow \quad & \mathcal{PE}_{P_1}(m'[n+1][1...\ell]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell]) \\ c[i] \quad \leftarrow \quad & (\mathcal{PE}_{P_1}(m''[i][1...\ell]) \\ \oplus \quad & \mathcal{PE}_{P_2}(c[i-1][1...\ell])) \end{aligned}$$

To re-encrypt, first the proxy produces $c'[i]$ using the following:

$$c'[i] \leftarrow \mathcal{PE}_{CK_1}(c[i] \oplus \mathcal{PE}_{P_2}(c[i-1][1...\ell]))$$

This step correctly transform the key P_1 to P_1', because:

$$\mathcal{PE}_{P_1}(m'[i][1...\ell]) = c[i] \oplus \mathcal{PE}_{P_2}(c[i-1][1...\ell])$$

and

$$\mathcal{PE}_{P_1'}(m'[i][1...\ell]) = \mathcal{PE}_{CK_1}(\mathcal{PE}_{P_1}(m'[i][1...\ell]))$$

Then, the proxy permutes the results using CK_3 to convert P_3 to P_3'

$$c''[1]...c''[n+1] \leftarrow \mathcal{PE}_{CK_3}(c'[1]...c'[n+1])$$

The proxy executes the following transformation to convert the key P_2 to P_2':

$$c[i] \leftarrow c[i]'' \oplus \mathcal{PE}_{P_2'}(c''[i-1][1...\ell])$$

These steps correctly convert the ciphertext $c[i]$ to new keys because:

$$
\begin{aligned}
c[i] &= \mathcal{PE}_{P'_1}(m'[i][1...\ell]) \oplus \mathcal{PE}_{P'_2}(c''[i-1][1...\ell]) \\
&= c[i]'' \oplus \mathcal{PE}_{P'_2}(c''[i-1][1...\ell])
\end{aligned}
$$

6.6 Security Analysis

6.6.1 Security Against Outsiders

First, we analyze the security of the encryption scheme against the attackers who have no access to any keys (outputs of $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{RG}$). In the Left-or-Right Indistinguishability, the adversary chooses two message blocks $M[0], M[1]$. The encryption oracle encrypts these blocks, and the adversary should distinguish whether the encrypted blocks belong to the left or right world.

Theorem 6.7. *Let F in $\mathcal{AONT}$ defined in Section 6.4.1 be a pseudorandom function and let $\mathcal{PR}$ be the symmetric encryption that supports proxy re-encryption defined in Section 6.5. The advantage of an adversary A attacking $\mathcal{PR}$ in LOR-CPA defined in Section 6.3 is at most:*

$$\mathsf{Adv}^{\mathsf{lor-cpa}}_{\mathcal{PR}}(A) \leq 2 \cdot \mathsf{Adv}^{prf}_{F}(B)$$

Proof. The complete proof is shown in Section 6.9.2. Basically, the proof is similar to the proof of the CTR mode. That is, the attacker cannot distinguish the ciphertext because to produce the ciphertext the plaintext is xor-ed with encrypted counters. The next processing (hashes and permutations) does not change the distribution of the ciphertext so the adversary cannot distinguish the outputs of encryption. □

We need to clarify that it seems without the permutations ($\mathcal{PE}$ in the encryption scheme) the outputs of AONT is indistinguishable to the random sequence of bits, so that it may be secure to be used as an encryption scheme without the permutations. This is partially true, because the characteristic of AONT's outputs is: if at least one of the outputs is missing, we cannot derive the original messages. Practically, if at least one block is missing, the original message is securely encrypted (with the key to decrypt is the missing block). However, if we have access to all blocks, it is easy to decrypt the blocks because we can derive the decryption key. The purpose of the permutations to the AONT' outputs is to remove access to the AONT' blocks, so that the attacker cannot derive the encryption key.

6.6.2 Security Against Previous Users

The previous users are the attackers who have access to previous outputs of key generations $\mathcal{G}_1$, $\mathcal{G}_2$ before re-encryption. The difference of the previous users to the normal attackers is they may also have access to all plaintext M_i (from previous accesses), intermediate values represented by AONT(M_i), and the key (K') used by AONT. However, the previous users cannot access any outputs of $\mathcal{RG}$ (which can only be accessed by the proxy).

Theorem 6.8. *Let F in $\mathcal{AONT}$ defined in Section 6.4.1 be a pseudorandom function and let $\mathcal{PR}$ be the symmetric encryption that supports proxy re-encryption defined in Section 6.5. The advantage of an adversary A, who have access to previous encryption keys, attacking $\mathcal{PR}$ in LOR-CPA defined in Section 6.3 is at most:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{lor-cpa}}_{\mathcal{PR}}(A) \leq \mathsf{Adv}^{prf}_{H}(B)\left(2+\frac{q(q-1)}{2^{\ell}}\right) \\ +\frac{q(q-1)}{2^{\ell}}\left(\frac{1}{2^{\ell}}+3(n+1)\right)\end{aligned}$$

where q is the number of queries, n is the number of blocks in each query, and ℓ is the size of each block.

Proof. The complete proof is shown in Section 6.9.3. The previous users have better advantage to the other attackers because they have access to the key (K') used in AONT function. To distinguish the ciphertext, the previous users try to find the collision on outputs of hashes on AONT function and also collision of the ciphertext by choosing the plaintext. The hash function limits the ability of the adversary to produce the collision on the ciphertext. □

6.6.3 Security Against Proxy

The proxy, that is the attacker who have access to the outputs of $\mathcal{RG}$, but he/she has no access to any outputs of $\mathcal{G}_1$, $\mathcal{G}_2$ and the intermediate values $\mathrm{AONT}(M_i)$. The data owner can delegate the re-encryption process to the proxy by providing the re-encryption keys (output of $\mathcal{RG}$) without the need to provide any access to the plaintext and encryption keys.

Theorem 6.9. *Let F in $\mathcal{AONT}$ defined in Section 6.4.1 be a pseudorandom function and let $\mathcal{PR}$ be the symmetric encryption that supports proxy re-encryption defined in Section 6.5. The advantages of an adversary A, who have access to the re-encryption keys (output of $\mathcal{RG}$), attacking $\mathcal{PR}$ in LOR-CPA defined in Section 6.3 is at most:*

$$\mathsf{Adv}^{\mathsf{lor-cpa}}_{\mathcal{PR}}(A) \leq 2 \cdot \mathsf{Adv}^{prf}_{F}(B)$$

Proof. The complete proof is shown in Section 6.9.4. The proof is similar to Theorem 6.7. The difference is the processing after CTR-style encryption and hashes uses less number of permutations. Because these processes does not change the distribution of the ciphertext, the adversary cannot distinguish the outputs of encryption better than distinguishing outputs of pseudorandom functions. □

6.6.4 A Note on Collusion Attack

It should be noted that we assume the proxy does not collude with any previous users who have access to the previous keys before encryption. It is easy to check that the proxy can recover the current encryption key if he/she has access to the previous keys. (However, as a comparison, even a costly double encryption method [141] is not resistant to the collusion attack).

6.7 Performance Evaluation

We analyze the performance of the scheme by calculating the computation costs to execute the scheme. Table 6.1 shows the computation costs that are needed by the scheme for the key generations ($\mathcal{G}_1, \mathcal{G}_2$), encryption ($\mathcal{E}$), decryption ($\mathcal{D}$), re-encryption key generation ($\mathcal{RG}$), and re-encryption ($\mathcal{RE}$). The computational costs are represented by the number of executions of the primitives for n number of blocks. The primitives are: AONT transform/de-transform ($\mathcal{AONT}$), permutation ($\mathcal{PE}$), de-permutation ($\mathcal{DP}$), finding conversion key ($\mathcal{FC}$), permutation key generation ($\mathcal{PG}$), random bits generation ($\xleftarrow{\$} \{0,1\}$) and XOR operation ($\oplus$).

Table 6.1: The number of primitive executions

Primitives	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{E}$	$\mathcal{D}$	$\mathcal{RG}$	$\mathcal{RE}$
$\mathcal{AONT}$	-	-	1	1	-	-
$\mathcal{PE}$	-	-	$2n+3$	-	-	$3n+3$
$\mathcal{DP}$	-	-	-	$2n+3$	-	-
$\mathcal{FC}$	-	-	-	-	2	-
$\mathcal{PG}$	-	3	3	3	6	2
$\xleftarrow{\$} \{0,1\}$	3	-	-	-	3	-
$\oplus$	-	-	$n+1$	$n+1$	-	$2n+2$

$\mathcal{PE}$, $\mathcal{DP}$, $\mathcal{FC}$, and XOR ($\oplus$) are cheap operations. The costs of AONT, $\mathcal{PG}$ and random generation are linear to a symmetric cipher operation. For example, the AONT scheme proposed by Rivest needs $2n$ encryption operations [144] while our AONT scheme needs roughly $3n$ encryption operations. As shown in Table 6.1, the costs of key generation and re-encryption key generation are linear to symmetric cipher operations because key generation needs 3 random bits generations while re-encryption keys generation needs 2 operations of find conversion key ($\mathcal{FC}$), 4 permutation key generations ($\mathcal{PG}$) and 4 random bits generations. The costs of encryption and decryption are also linear to symmetric cipher operations because encryption needs execution of an AONT transform, $2n+3$ permutations, 3 permutation key generations, and n times XOR, while decryption needs execution of an AONT transform, $2n+3$ de-permutations, 3 permutation key generations, and n times XOR. Re-encryption in this scheme is very efficient because it only needs 2 permutation key generations regardless of the number of blocks and a linear number of cheap $\mathcal{PE}$ and XOR operations ($3n+3$ $\mathcal{PE}$ and $2n+2$ XOR operations).

6.8 Discussion: Using CBC and CTR modes as Alternatives to AONT

The security of the scheme in Section 6.5 relies on the difficulty to find the correct position of the AONT transformation. Other encryption modes (CBC and CTR) also have characteristics that the blocks cannot be decrypted correctly if we cannot find the correct position of the blocks. In CTR, we should know the correct position of a block for decrypting the block. In CBC mode, we should know a pair of consecutive blocks to decrypt a block. It is interesting to know whether it is possible to implement the scheme showed in Section 6.5 using CBC or CTR.

6.8.1 Using CBC mode

Each block encrypted with CBC is defined as $c_i = E(k, p_i \oplus c_{i-1}), c_0 = IV$, where IV is an initialization vector [82]. To decrypt a block c_i we use the following formula: $p_i = E(k, c_i) \oplus c_{i-1}, c_0 = IV$. If we substitute the AONT with a CBC mode there are some features of AONT that cannot be substituted by CBC:

1. In CBC, we only need to know two consecutive modes and decrypt a block to get a block of the plaintext, while in the AONT we need to know all blocks with correct position and decrypt all blocks to get any block of the plaintext.
2. In CBC, we do not need to have all of blocks, we can decrypt any blocks and get a block of plaintext by knowing two consecutive blocks.

Those characteristics of CBC affects the security of the re-encryption as follows:

- the previous user does not need to know all bits key P_3 to decrypt a part of the blocks. By using CBC, it is possible the previous user shares a part of the blocks without knowing all bits of the key (which is not possible in All-or-Nothing-Transform (AONT), because in AONT to leak a block, we need to have all correct bits of the key).
- for the proxy and the normal attackers, by using CBC mode, attack to find the encryption key is easier, because the proxy and the outsider can check the correctness of a key for a block by only decrypting the block rather than decrypting all blocks of the message with correct keys for all blocks in AONT scheme to test the correctness of any bits of key in a block.

6.8.2 Using CTR mode

In CTR, we use a nonce n and counter i to encrypt each block of the message. Each block encrypted with CTR is defined as $c_i = E(k, n \oplus i) \oplus p_i$ [82]. To decrypt a block c_i we use the following formula: $p_i = E(k, n \oplus i) \oplus c_i$. The CTR mode has the same weaknesses as the CBC so that if we use CTR there are some advantages for the previous users, the proxy and the outsiders:

- the previous user does not need to know all bits key P_3 to decrypt a part of the blocks. By using CTR, it is possible the previous user shares a part of the blocks without knowing all bits of the key (which is not possible in scheme that uses AONT).
- attack to find the encryption key is easier for the proxy and the normal attackers if we use CTR, because the proxy and the outsider are able to check the correctness of a key (for the block) by only decrypting a block of message rather than decrypting all blocks with the correct key in AONT scheme.

6.9 Proof of the Theorems

In this section, we show the proofs of the theorems by using the game-based proof strategy as described in [99, 153]. In the game-based proof, a security proof is represented by a sequence of games **G0** to **GN** where the first game (**G0**) is the original chosen-plaintext attack game, while the last game (**GN**) is a target the game with a computable probability. We manipulate each game by transforming the game into another game, and compute the difference probabilities of successful attacks in the consecutive games.

We need to clarify the notations we use in each game in our proofs. In our proofs, each game consists of q queries asked by the attacker, that adaptively chooses the messages to be converted by the oracle (an AONTH oracle or an encryption oracle). The oracle uses the inputs from the attacker, a random value b, and the security parameters (i.e., encryption keys) to generate responds to the attacker. At the end of each game, after receiving responds from the oracle, the adversary should guess the value of b. We represent the game in a simplified programming language that uses some of the notations as follows:

1. $\leftarrow$, output of, for example in $a \leftarrow f(b)$, a is the output of $f(b)$.

2. $\xleftarrow{\$}$ represents the output of a random function. For example $K' \xleftarrow{\$} \{0,1\}^\ell$ says that the variable K' is an output of a random function with length ℓ

3. $\oplus$, binary exclusive OR (XOR), for example $a \oplus b$ a XOR b

4. [], an element of an array, for example $m_i[1]$ represents the first element of the array m_i

6.9.1 Proof of Theorem 6.6

Game based proof.

G0. Game **G0** represents the original game. In each query i, the adversary A chooses n blocks plaintexts (M_i) and a value s_i where $1 \leq s \leq n+1$ and given access to the AONT oracle. The oracle transforms the plaintexts and return the pseudomessages with one missing block and also the challenge block $m'_i[s_i]$, the adversary should guess whether the challenge block belong to the pseudomessage or a random sequence of bits. To provide the unique *ctr* for each AONTH call, we use $i \cdot n$ as the *ctr* where i is the query number and n is the number of blocks in each query.

G0 - Theorem 6.6

```
$K' \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i, s_i) \leftarrow A(M'_1, ..., M'_{i-1})$
    $m_i[1], ..., m_i[n] \leftarrow M_i$
    for $j = 1$ to $n$ do
        $\boxed{x[j] \leftarrow m_i[j] \oplus F_{K'}(i \cdot n + j)}$
    end for
    $\boxed{m'_i[n+1] \leftarrow K' \oplus H(x[1] \cdots x[n])}$
    for $j = 1$ to $n$ do
        $\boxed{m'_i[j] \leftarrow x[j] \oplus H(m'_i[n+1] \oplus (i \cdot n + j))}$
    end for
    $b \xleftarrow{\$} \{0,1\}$
    if $b = 1$ then
        $m'_i[s_i] \xleftarrow{\$} \{0,1\}^\ell$
    end if
    $M'_i \leftarrow m'_i[1], ..., m'_i[n+1]$
end for
$d \leftarrow A(M'_1, ..., M'_q)$
return $(b = d)$
```

The advantages of the adversaries is defined as:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{AO}}^{aon-cpa-b}(A) &= 2 \cdot \Pr\left[\mathbf{Exp}_{\mathcal{AO},A_{cpa}}^{aon-cpa}(b) = b\right] - 1 \\ &= 2 \cdot \Pr\left[G_0^A \Rightarrow 1\right] - 1\end{aligned}$$

G1 - Theorem 6.6

```
$K' \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i, s_i) \leftarrow A(M'_1, ..., M'_{i-1})$
    $m_i[1], ..., m_i[n] \leftarrow M_i$
    for $j = 1$ to $n$ do
        $\boxed{x[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    $\boxed{m'_i[n+1] \xleftarrow{\$} \{0,1\}^\ell}$
    for $j = 1$ to $n$ do
        $\boxed{m'_i[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    if $b = 1$ then
        $m'_i[s_i] \xleftarrow{\$} \{0,1\}^\ell$
    end if
    $M'_i \leftarrow m'_i[1], ..., m'_i[n+1]$
end for
$d \leftarrow A(M'_1, ..., M'_q)$
return $(b = d)$
```

G1. In this game, we assume F and H are pseudorandom functions with a PRF advantage $\mathrm{Adv}_F^{prf}(B)$, so that:

$$\Pr[G_0^A \Rightarrow 1] = \Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)$$

The above equation says that the advantage of the adversary to distinguish the values of $K' \oplus H(x[1] \cdots x[n])$ from the random values and the values of $x[j] \oplus H(m'_i[n+1] \oplus (i \cdot n + j))$ from random values where $x[j] = m_i[j] \oplus F_{K'}(i \cdot n + j)$ and K' is a random value should be less than $\mathsf{Adv}_F^{prf}(B)$. This result can be derived directly from the Definition 6.3.

Because in game **G1** the ciphertext is produced randomly, then

$$\Pr[G_1^A \Rightarrow 1] = \frac{1}{2}$$

and

$$\begin{aligned}\mathbf{Adv}_{\mathcal{AO}}^{aon-cpa-b}(A) &= 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1 \\ &\leq 2 \cdot \Big(\Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)\Big) - 1 \\ &\leq 2 \cdot \mathsf{Adv}_F^{prf}(B) + 2 \cdot \frac{1}{2} - 1 \\ &\leq 2 \cdot \mathsf{Adv}_F^{prf}(B)\end{aligned}$$

6.9.2 Proof of Theorem 6.7

Game based proof.

G0. This game represents the original game. In each query i, the adversary A chooses two n blocks plaintexts $(M_i[0], M_i[1])$ and given access to the encryption oracle. The oracle encrypts the plaintexts and returns the ciphertext, the adversary should guess whether the ciphertexts belong to the left (0) or right (1).

G0 - Theorem 6.7

```
$P_1 \xleftarrow{\$} P, P_2 \xleftarrow{\$} P, P_3 \xleftarrow{\$} P$
$b \xleftarrow{\$} \{0,1\}, K' \xleftarrow{\$} \{0,1\}^\ell, iv \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(C_1, ..., C_{i-1})$ // if $i > 1$ the attacker can access his chosen plaintext-ciphertext pairs from the previous requests
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    for $j = 1$ to $n$ do
        $\boxed{x[j] \leftarrow m[j]_i \oplus F_{K'}(ctr + j)}$
    end for
    $\boxed{m_i'[n+1] \leftarrow K' \oplus H(x[1]...x[n])}$
    for $j = 1$ to $n$ do
        $\boxed{m_i'[j] \leftarrow x[j] \oplus H(m_i'[n+1] \oplus (ctr + j))}$
    end for
    $\boxed{c_i[0] \leftarrow \mathcal{PE}_{P_1}(m_i'[n+1]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])}$
    for $j = 1$ to $n$ do
        $\boxed{c_i[j] \leftarrow \mathcal{PE}_{P_1}(m_i'[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])}$ // we represent permutation of blocks with key $P_3$ in encryption function as $m_i'[P_3[j]]$
    end for
    $C_i \leftarrow c_i[0]...c_i[n]$
end for
$d \leftarrow A(C_1, ..., C_q)$
return $(b = d)$
```

As shown in the **G0**, the values of P_1, P_2, P_3, b, K', and iv are fixed during the game because they are chosen before the attackers deciding the chosen messages. In the LOR-CPA, the oracle encrypts $M_i[0]$ or $M_i[1]$ based on the random bit value b. The value of b is not revealed until the attacker has decided his/her guess. Let $\Pr[G_0^A \Rightarrow 1]$ be the probability that the adversary correctly guesses the left or right oracle, as shown in Section 6.3 then:

$$\mathbf{Adv}_{\mathcal{PR}}^{lor-cpa-b}(A) = 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1$$

G1. In game **G1**, we change the outputs of each F and H to random values, and by definition of the PRF advantage, the difference of the adversary's advantage in **G0** and **G1** is at most $\mathsf{Adv}_F^{prf}(B)$:

$$\Pr[G_0^A \Rightarrow 1] = \Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)$$

Because the input to H is random, the outputs of H are also randoms. And, because the inputs to permutations $c_i[0] \leftarrow \mathcal{PE}_{P_1}(m_i'[n+1]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])$ and $c_i[j] \leftarrow \mathcal{PE}_{P_1}(m_i'[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])$ are random, those permutations will also produce random sequence of blocks. To distinguish the ciphertext, the attackers should find a collision in the outputs $c_i[j]$, otherwise the attacker does not have better information than the random outputs.

G1 - Theorem 6.7

```
$P_1 \xleftarrow{\$} P, P_2 \xleftarrow{\$} P, P_3 \xleftarrow{\$} P$
$b \xleftarrow{\$} \{0,1\}, K' \xleftarrow{\$} \{0,1\}^\ell, iv \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    for $j = 1$ to $n$ do
        $\boxed{x[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    $\boxed{m_i'[n+1] \xleftarrow{\$} \{0,1\}^\ell}$
    for $j = 1$ to $n$ do
        $\boxed{m_i'[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    $\boxed{c_i[0] \leftarrow \mathcal{PE}_{P_1}(m_i'[n+1]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])}$
    for $j = 1$ to $n$ do
        $\boxed{c_i[j] \leftarrow \mathcal{PE}_{P_1}(m_i'[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])}$
    end for
    $C_i \leftarrow c_i[0]...c_i[n]$
end for
$d \leftarrow A(C_1, ..., C_q)$
return $(b = d)$
```

Because in game **G1** the ciphertext is now produced randomly, then the probability of the attacker successfully guess b is $\frac{1}{2}$.

$$\Pr[G_1^A \Rightarrow 1] = \frac{1}{2}$$

and

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{PR}}^{lor-cpa}(A) = 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1 \\
&\leq 2 \cdot \left(\Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)\right) - 1 \\
&\leq 2 \cdot \mathsf{Adv}_F^{prf}(B) + 2 \cdot \frac{1}{2} - 1 \leq 2 \cdot \mathsf{Adv}_F^{prf}(B)
\end{aligned}$$

6.9.3 Proof of Theorem 6.8

G0/G1. We need to analyze the possibilities of distinguishing the ciphertext for the second types of the attacker. We identify that there are two possibilities that the attacker may distinguish the ciphertext:

(a) by fabricating the desired ciphertexts, or

(b) by finding the collisions on the ciphertexts.

In the game **G0**, we assume that the adversary can also choose $x[j]$ along with the chosen block $m[j]$ because it knows the encryption key K' used by F, but the adversary cannot easily choose $m'[j]$, otherwise the attacker should break the security of the hash function H. To attack the scheme, first the adversary tries to fabricate the ciphertext $c_i[0]$ by finding three plaintexts $x_i[1], ..., x_i[n]$, $x_k[1], ..., x_k[n]$, $x_{k'}[1], ..., x_{k'}[n]$ where $H(x_i[1], ..., x_i[n]) = H(x_k[1], ..., x_k[n]) \oplus H(x_{k'}[1], ..., x_{k'}[n])$ and three $iv_i, iv_k, iv_{k'}$ where $iv_i = iv_k \oplus iv_{k'}$, then the adversary can compute $m'_i[n+1]$ and distinguish $c_i[0]$, because

$$m'_i[n+1] = m'_k[n+1] \oplus m'_{k'}[n+1] \oplus K'$$

for a fixed K', $k < i$, and $k' < i$.

```
G0 - Theorem 6.8
$P_1 \xleftarrow{\$} P, K_2 \xleftarrow{\$} \{0,1\}^l, P_2 \xleftarrow{\$} P, P_3 \xleftarrow{\$} P$
$b \xleftarrow{\$} \{0,1\}, iv \xleftarrow{\$} \{0,1\}^\ell, K' \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(C_1, ..., C_{i-1})$// if $i > 1$ the attacker can access his chosen plaintext-ciphertext pairs from the previous requests
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    for $j = 1$ to $n$ do
        $x[j] \leftarrow \{0,1\}^\ell$ // the adversary can choose $x[j]$, and
        $m_i[j] \leftarrow x[j] \oplus F_{K'}(ctr + j)$ // compute the associated $m[j]$
    end for
    $\boxed{m'_i[n+1] \leftarrow K' \oplus H(x[1]...x[n])}$
    for $j = 1$ to $n$ do
        $\boxed{m'_i[j] \leftarrow x[j] \oplus H(m'_i[n+1] \oplus (ctr + j))}$
    end for
    $\boxed{c_i[0] \leftarrow \mathcal{PE}_{P_1}(m'_i[n+1]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])}$
    for $j = 1$ to $n$ do
        $\boxed{c_i[j] \leftarrow \mathcal{PE}_{P_1}(m'_i[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])}$
    end for
    $C_i \leftarrow c_i[0]...c_i[n]$
end for
$d \leftarrow A(C_1, ..., C_q)$
return $(b = d)$
```

We argue that the probability of such condition is at most $\frac{q(q-1)}{2^{\ell+1}} \cdot \left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right)$. At first, we need to find the probability of $iv_i = iv_k \oplus iv_{k'}$ for randoms $iv_i, iv_k, iv_{k'}$. The probability of such condition is at most $\frac{q(q-1)}{2^{\ell+1}}$, because the XOR or two randoms $(iv_k \oplus iv_{k'})$ produces another random value, so the possibility of finding $iv_i = iv_k \oplus iv_{k'}$ is at most the same as finding collision of two random outputs which is at most $\frac{q(q-1)}{2^{\ell+1}}$.

The advantage of an adversary to find three plaintexts $x_i[1], ..., x_i[n]$, $x_k[1], ..., x_k[n]$, $x_{k'}[1], ..., x_{k'}[n]$ where $H(x_i[1], ..., x_i[n]) = H(x_k[1], ..., x_k[n]) \oplus H(x_{k'}[1], ..., x_{k'}[n])$ should be less than $\mathbf{Adv}_H^{prf}(B)$, because if the adversary can distinguish the outputs of H from the random outputs (i.e., guessing the outputs of H), it can easily fabricate these three plaintexts.

Alternatively, rather than finding the three plaintexts, the attacker may guess $m'_i[n+1]$ directly with the probability $\frac{1}{2^\ell}$. So that, the probability of all of the above conditions is at most $\frac{q(q-1)}{2^{\ell+1}} \cdot \left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right)$.

If in game **G1**, we assume that the attacker fails fabricating $c_i[0]$ and also fails guessing $m'_i[n+1]$, then:

$$\Pr[G_0^A \Rightarrow 1] - \Pr[G_1^A \Rightarrow 1] \leq \frac{q(q-1)}{2^{\ell+1}} \left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right)$$

G2. Because $c_i[j] \leftarrow \mathcal{PE}_{P_1}(m'_i[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])$, to distinguish the ciphertext, in each query i, the adversary needs to find the collision $c_k[j] = c_i[j'](0 \leq j \leq n)$, so that he/she can deduce $\mathcal{PE}_{P_1}(m'_k[P_3[j+1]] \oplus m'_i[P_3[j'+1]])$. This information can be used to deduce P_1 by checking the number of 1s or 0s bits of $m'_k[P_3[j+1]] \oplus m'_i[P_3[j'+1]]$. The possibility of collision is defined in the event "bad".

G2 - Theorem 6.8

```
$P_1 \xleftarrow{\$} P, K_2 \xleftarrow{\$} \{0,1\}^l, P_2 \xleftarrow{\$} P, P_3 \xleftarrow{\$} P$
$b \xleftarrow{\$} \{0,1\}, iv \xleftarrow{\$} \{0,1\}^\ell, K' \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(C_1, ..., C_{i-1})$
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    for $j = 1$ to $n$ do
        $x[j] \leftarrow \{0,1\}^\ell$ // the adversary can choose $x[j]$, and
        $m_i[j] \leftarrow x[j] \oplus F_{K'}(ctr + j)$ // compute the associated $m[j]$
    end for
    $H_{i,0} \xleftarrow{\$} \{0,1\}^\ell$
    $\boxed{m'_i[n+1] \leftarrow K' \oplus H_{i,0}}$
    for $j = 1$ to $n$ do
        $H_{i,j} \xleftarrow{\$} \{0,1\}^l$
        $\boxed{m'_i[j] \leftarrow x[j] \oplus H_{i,j}}$
    end for
    $\boxed{c_i[0] \leftarrow \mathcal{PE}_{P_1}(m'_i[n+1]) \oplus \mathcal{PE}_{P_2}(iv[1...\ell])}$
    if $c_i[0] \in S$ then
        bad ← true
    end if
    $S \leftarrow S \cup \{c_i[0]\}$
    for $j = 1$ to $n$ do
        $\boxed{c_i[j] \leftarrow \mathcal{PE}_{P_1}(m'_i[P_3[j]]) \oplus \mathcal{PE}_{P_2}(c_i[j-1])}$
        if $c_i[j] \in S$ then
            bad ← true
        end if
        $S \leftarrow S \cup \{c_i[j]\}$
    end for
    $C_i \leftarrow c_i[0], ..., c_i[n]$
end for
$d \leftarrow A(C_1, ..., C_q)$
return $(b = d)$
```

In game **G2**, we change the outputs of H to random outputs so that

$$\Pr[G_2^A \Rightarrow 1] = \frac{1}{2}$$

and

$$\Pr[G_1^A \Rightarrow 1] - \Pr[G_2^A \Rightarrow 1] \leq \mathbf{Adv}_H^{prf}(B) + \Pr[G_2^A \text{sets bad}]$$

Because each $m'_i[j]$ is produced randomly and $\mathcal{PE}$ is a permutation, the probability of collision is the total probability of bad events, that is the collision of $m'_i[n+1], iv_i, m'_i[n+1] \oplus iv_i, m'_i[j], c_j$ and $m'_i[j] \oplus c_j$. Because for each query, the algorithm produces three blocks $m'_i[n+1], iv_i, m'_i[n+1] \oplus iv_i$ and $3n$ blocks $m'_i[j], c_j, m'_i[j] \oplus c_j$ totaling $3(n+1)$ blocks, the probability of collision is:

$$\begin{aligned}
&\Pr[G_2^A \text{sets bad}] \\
&\leq \frac{3(n+1) + 2 \cdot 3(n+1) + ... + (q-1) \cdot 3(n+1)}{2^\ell} \\
&\leq 3(n+1)\frac{q(q-1)}{2^{\ell+1}}
\end{aligned}$$

So, that

$$\Pr[G_1^A \Rightarrow 1] - \Pr[G_2^A \Rightarrow 1] \leq \mathbf{Adv}_H^{prf}(B) + 3(n+1)\frac{q(q-1)}{2^{\ell+1}}$$

And,

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{PR}}^{lor-cpa}(A) = 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1 \\
&\leq 2 \cdot \left(\Pr[G_1^A \Rightarrow 1] + \frac{q(q-1)}{2^{\ell+1}}\left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right)\right) - 1 \\
&\leq 2 \cdot \frac{q(q-1)}{2^{\ell+1}}\left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right) \\
&+ 2 \cdot \left(\mathbf{Adv}_H^{prf}(B) + 3(n+1)\frac{q(q-1)}{2^{\ell+1}} + \Pr[G_2^A \Rightarrow 1]\right) - 1 \\
&\leq 2 \cdot \frac{q(q-1)}{2^{\ell+1}}\left(\mathbf{Adv}_H^{prf}(B) + \frac{1}{2^\ell}\right) \\
&+ 2 \cdot \left(\mathbf{Adv}_H^{prf}(B) + 3(n+1)\frac{q(q-1)}{2^{\ell+1}}\right) + 2 \cdot \frac{1}{2} - 1 \\
&\leq \mathbf{Adv}_H^{prf}(B)\left(2 + \frac{q(q-1)}{2^\ell}\right) + \frac{q(q-1)}{2^\ell}\left(\frac{1}{2^\ell} + 3(n+1)\right)
\end{aligned}$$

6.9.4 Proof of Theorem 6.9

G0. In **G0** is similar with the proof in Theorem 6.7.

G1. In **G1**, F produces random outputs, and we remove $\mathcal{PE}_{P_2}$ because the proxy knows the keys P_2 and P_2'.

```
**G1** - Theorem 6.9
$P_1 \xleftarrow{\$} P, P_2 \xleftarrow{\$} P, P_3 \xleftarrow{\$} P$
$b \xleftarrow{\$} \{0,1\}, K' \xleftarrow{\$} \{0,1\}^\ell, iv \xleftarrow{\$} \{0,1\}^\ell$
for $i \leftarrow 1...q$ do
    $(M_i[0], M_i[1]) \leftarrow A(C_1, ..., C_{i-1})$// if $i > 1$ the attacker can access his chosen plaintext-ciphertext pairs from the previous requests
    $m_i[1]...m_i[n] \leftarrow M_i[b]$
    for $j = 1$ to $n$ do
        $\boxed{x[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    $\boxed{m_i'[n+1] \xleftarrow{\$} \{0,1\}^\ell}$
    for $j = 1$ to $n$ do
        $\boxed{m_i'[j] \xleftarrow{\$} \{0,1\}^\ell}$
    end for
    $\boxed{c_i[0] \leftarrow \mathcal{PE}_{P_1}(m_i'[n+1])}$
    for $j = 1$ to $n$ do
        $\boxed{c_i[j] \leftarrow \mathcal{PE}_{P_1}(m_i'[P_3[j]])}$
    end for
    $C_i \leftarrow c_i[0]...c_i[n]$
end for
$d \leftarrow A(C_1, ..., C_q)$
return $(b = d)$
```

Because now the ciphertext is produced randomly, then

$$\Pr[G_1^A \Rightarrow 1] = \frac{1}{2}$$

and

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{PR}}^{lor-cpa}(A) = 2 \cdot \Pr[G_0^A \Rightarrow 1] - 1 \\
&\leq 2 \cdot \left(\Pr[G_1^A \Rightarrow 1] + \mathsf{Adv}_F^{prf}(B)\right) - 1 \\
&\leq 2 \cdot \mathsf{Adv}_F^{prf}(B) + 2 \cdot \frac{1}{2} - 1 \leq 2 \cdot \mathsf{Adv}_F^{prf}(B)
\end{aligned}$$

6.10 Discussion: An Attempt to Develop a Secure Proxy Re-encryption Using Pure Symmetric Cipher

A proxy re-encryption cannot be implemented using currently used practical one-way functions (i.e., SHA and AES). We show the argument by trying to develop a proxy re-encryption for symmetric cipher using Xor-scheme (Vernam cipher), stream cipher and block cipher.

6.10.1 Xor-scheme (Vernam Cipher)

In the Vernam Cipher, the key is generated by random function with the same length of the message. Besides its big key size, it also does not support secure proxy cryptography.

Let K_A, K_B be two random keys and E is the encryption function, then we can produce the ciphertext C_A and C_B using the following equations:

$$C_A = E(K_A, M) = K_A \oplus M \tag{6.1}$$

$$C_B = E(K_B, M) = K_B \oplus M \tag{6.2}$$

To support the proxy re-encryption, there is a function P that converts C_A to C_B where:

$$C_B = P(E(K_A, M) = P(K_A \oplus M)$$

This equation is fulfilled by $P = (K_A \oplus K_B)\oplus$, because

$$C_B = (K_A \oplus K_B) \oplus (K_A \oplus M) = K_B \oplus M$$

However, this method is not secure because from the equation 6.1 and 6.2:

$$\begin{aligned} M = K_A \oplus C_A &= K_B \oplus C_B \\ K_B &= K_A \oplus C_A \oplus C_B \end{aligned} \tag{6.3}$$

In the equation 6.3, the user who knows the previous ciphertext C_A, previous key K_A and the current ciphertext C_B can easily compute K_B.

6.10.2 Stream cipher-like encryption

In the stream cipher, the plaintext $(m_0, m_1, ..., m_{n-1})$ is encrypted to produce the ciphertext $(c_0^A, c_1^A, ..., c_{n-1}^A)$ using a random generator R with a private key K_A as follows:

$$c_i^A = R(K_A, m_{i-1}, ...m_0, c_{i-1}^A, ...c_0^A) \oplus m_i \tag{6.4}$$

To implement the proxy re-encryption that converts the ciphertext encrypted with key K_A to the ciphertext encrypted with key K_B, the proxy should be able to re-encrypt the ciphertext c_i^A into the ciphertext

$$c_i^B = R(K_B, m_{i-1}, ...m_0, c_{i-1}^B, ...c_0^B) \oplus m_i \tag{6.5}$$

using a conversion function P, where

$$\begin{aligned} c_i^B &= P(c_0^A, c_1^A, ..., c_{n-1}^A) \\ &= P(R(K_A) \oplus m_0, R(K_A, m_0, c_0^A) \oplus m_1, ... \\ &\quad R(K_A, m_{n-2}, ...m_0, c_{n-2}^A, ...c_0^A) \oplus m_{n-1}) \end{aligned}$$

so that:

$$\begin{aligned}
R(K_B, m_{i-1}, ...m_0, c_{i-1}^B, ...c_0^B) \oplus m_i &= P(R(K_A) \oplus m_0, R(K_A, m_0, c_0^A) \oplus m_1, ... \\
&\quad R(K_A, m_{n-2}, ...m_0, c_{n-2}^B, ...c_0^B) \oplus m_{n-1}) \\
R(K_B, m_{i-1}, ...m_0, c_{i-1}^A, ...c_0^A) &= P(R(K_A) \oplus m_0, R(K_A, m_0, c_0^A) \oplus m_1, ... \\
&\quad R(K_A, m_{n-2}, ...m_0, c_{n-2}^A, ...c_0^A) \\
&\quad \oplus m_{n-1}) \oplus m_i \qquad (6.6)
\end{aligned}$$

The users who are previously granted access to the message m should not be able to infer K_B even if the users know the previous key K_A, all plaintexts $(m_0, m_1, ..., m_{n-1})$ and all ciphertexts $(c_0^A, c_1^A, ..., c_{n-1^A}, c_0^B, c_1^B, ..., c_{n-1}^B)$. There are two possibilities:

1. (A strict requirement): the proxy can collude with the user, so that the users know P from the proxy.

   If the proxy colludes, the user can compute the right side of the equation 6.6. For instance the result is A, then

$$\begin{aligned}
P(R(K_A) \oplus m_0, R(K_A, m_0, c_0^A) \oplus m_1, ... & \\
R(K_A, m_{n-2}, ...m_0, c_{n-2}^A, ...c_0^A) \oplus m_{n-1}) \oplus m_i &= A
\end{aligned}$$

   and

$$R(K_B, m_{i-1}, ...m_0, c_{i-1}^B, ...c_0^B) = A \qquad (6.7)$$

   Because the user knows all plaintexts and ciphertexts (he/she only does not have K_B), for a secure proxy re-encryption scheme it should not be possible to calculate K_B. This requirement is only fulfilled if the function R is a one-way function so it is not possible to infer K_B from the output of R and some others inputs.

2. (A less strict requirement): the proxy does not collude with the user, so that the users does not know P.

From equations 6.4 and 6.5 we get:

$$\begin{aligned} m_i &= R(K_A, m_{i-1}, ...m_0, c^A_{i-1}, ...c^A_0) \oplus c^A_i \\ m_i &= R(K_B, m_{i-1}, ...m_0, c^B_{i-1}, ...c^B_0) \oplus c^B_i \end{aligned}$$

then

$$R(K_B, m_{i-1}, ...m_0, c^B_{i-1}, ...c^B_0) = R(K_A, m_{i-1}, ...m_0, c^A_{i-1}, ...c^A_0) \oplus c^A_i \oplus c^B_i$$

Because the user can compute the right side (for instance the result is B), we get the formula similar to equation 6.7.

$$R(K_B, m_{i-1}, ...m_0, c^B_{i-1}, ...c^B_0) = B \tag{6.8}$$

With the same reason as the equation 6.7, for a secure proxy re-encryption scheme, the function R should be a one-way function that protect K_B even if the users know the outputs and other inputs.

If R is a one-way function, from the equation 6.6 we have the following equations:

$$\begin{aligned} R(K_B, m_{i-1}, ...m_0, c^B_{i-1}, ...c_0)^B &= P(R(K_A) \oplus m_0, R(K_A, m_0, c^A_0) \oplus m_1, ... \\ &\quad R(K_A, m_{n-2}, ...m_0, c^A_{n-2}, ...c^A_0) \\ &\quad \oplus m_{n-1}) \oplus m_i \\ R(K_B, m_i, ...m_0, c^B_i, ...c^B_0) &= P(R(K_A) \oplus m_0, R(K_A, m_0, c^A_0) \oplus m_1, ... \\ &\quad R(K_A, m_{n-2}, ...m_0, c^A_{n-2}, ...c^A_0) \\ &\quad \oplus m_{n-1}) \oplus m_{i+1} \\ &\vdots \qquad\qquad \vdots \end{aligned}$$

From the above equations we can conclude that to have a proxy cryptography, we should find a one-way function which we can do operation to the ciphertexts meaningfully.

We can simplify the above equation (i.e the random function R only uses the private key K_A or K_B and a counter i) to the following equations:

$$\begin{array}{rcl} R(K_A, i) & = & P(R(K_B, i) \oplus m_i) \\ R(K_A, i+1) & = & P(R(K_B, i+1) \oplus m_{i+1}) \\ \vdots & & \vdots \end{array}$$

With this simplification, we still need a special one-way function which the ciphertexts are easily converted to another ciphertext with different keys. This property is undesirable and not possible in the current practical one-way function used in the real world (i.e., SHA or AES-based one-way functions).

6.10.3 Block cipher-like encryption

In the block cipher-like encryption, the plaintext $(m_0, m_1, ..., m_{n-1})$ is encrypted to produce the ciphertext $(c_0^A, c_1^A, ..., c_{n-1}^A)$ using a block cipher encryption E with a key K_A as follows:

$$c_i^A = E(K_A, m_i, m_{i-1}, ...m_0, c_{i-1}^A, ...c_0^A) \tag{6.9}$$

To implement the proxy re-encryption, the proxy should be able to convert the ciphertext c_i^A encrypted with the private key K_A into another ciphertext c_i^B encrypted with the key K_B as follows:

$$c_i^B = E(K_B, m_i, m_{i-1}, ...m_0, c_{i-1}^B, ...c_0^B) \tag{6.10}$$

With the same arguments as the stream cipher in Section 6.10.2, the E should be a one-way function and has properties that cannot be implemented with the current practical one-way functions.

6.11 Conclusion

In this chapter, we have showed a secure proxy re-encryption scheme for the symmetric cipher. The main advantage of our scheme is that we can update the key of the encrypted data faster than decrypt-and-then-encrypt method. It is very useful in the application of data encryption in offline or online databases.

Chapter 7

Conclusion

We conclude this thesis by pointing out our important results as follows:

1. **The Integrity Scheme for the Provenance Recording System**. We have proposed a combination of using the signature chain and labeling each provenance assertion with a trusted counter to protect the integrity of the assertions in the provenance recording systems. Our method can detect the integrity attacks that cause the problems namely "inconsistent claims and interpretation" problems.

2. **The Access Control Method for the Provenance Recording System**. We have proposed an access control method that is suitable to be implemented in a provenance recording system. Our method allows restricting access to the provenance assertions by considering the relationships between the processes and data.

3. **The Signature Scheme for a Sequence of Digital Documents**. We proposed a signature scheme for a sequence of digital documents that uses signature with message recovery as the primitive. The main characteristics of the signature scheme is we can verify the integrity of a member of the sequence without accessing the other members of the sequence.

4. **The Proxy Re-encryption Scheme for Symmetric Cryptography**. We developed a proxy re-encryption method for the symmetric key cryptography by first converting the message to a pseudomessage using an All-or-Nothing Transform (AONT) and converting the pseudomessage using some simple permutations so that the ciphertext can be transformed efficiently.

In the following, we survey recent research results on the provenance and signature-chain and also research on the proxy re-encryption. We also suggest some possible future work.

7.1 Recent Research on the Provenance and the Signature Chain

Security and applications of the provenance are active research topics. Bani Taha et al. proposed trusted tamper-evident provenance by using Trusted Platform Module (TPM). Basically the idea is by storing the hash values in the TPM [154]. Chen et al. proposed access control to the provenance graph by controlling access to the provenance sub-graph [155]. Xie et al. developed Intrusion Detection System (IDS) by analyzing the provenance collected in PASS [156]. Yap et al. developed method for attestation (checking the trustworthiness of a system) by checking the provenance generated in the system [157]. Chen et al. proposed diagnostic method for scientific workflow by using the provenance [158]. McClatchey et al. discussed the traceability through the provenance in the CRISTAL: a big data for medical system [159]. Jamil et al. proposed the method to secure the provenance by using authenticated data structure approach [160].

Recently, the provenance and the signature chain found their application in the financial industry in the form of block-chain technology. The block chain is basically a form of signature and hash chain that is used to protect the integrity of the transaction data. An implementation of the block-chain (i.e., bitcoin) is used to implement a popular crypto-currency [65]. The main feature of the bitcoin protocol is that it does not require any trusted party to decide which transaction should be recorded in the chain. The bitcoin protocol is rather using a consensus protocol where the transaction can only be recorded by a party that successfully computes a specific form of hash value. The weaknesses of the bitcoin protocol are it can only process a limited transaction in a second and it needs to execute a costly computation to find the specific form of the hash value.

The block-chain technology can also be used to record other data securely, so it may find applications in other fields (i.e., supply chain, academic record, medical record, law (contracts), etc). Kosba et al. proposed the block-chain model for smart contracts [161], that is followed by many other models [162–165]. Weber et al. applied the block-chain in the process monitoring [166, 167]. Abeyratne et al. surveyed the applications of block-chain in many fields especially manufacturing supply chain [168] while Wu et al. proposed a specific implementation of the block-chain technology to protect the supply chain [169]. The other researchers used the block-chain to protect the medical data [170–173], and tried to apply the block-chain technology to secure the academic/educational [174] and industrial data [175].

7.2 Recent Research on the Proxy Re-encryption

Most recent research in proxy re-encryption are mainly focusing on public key versions using bilinear maps (either an id-based or a non-id-based) and their applications in the clouds. Lu et al. [176] proposed pairing-free proxy re-encryption scheme based on Schnorr's signature scheme [91] and Fujisaki/Okamoto's hash-enhanced ElGamal public-key encryption scheme [177]. Canard et al. applied the proxy re-encryption scheme to protect the privacy of tree-structured data in an untrusted storage provider [178]. Shao et al. proposed another bidirectional proxy re-encryption [179]. Ohata et al. proposed the method to verify the results of re-encryption by the proxy [180], while Peng et al. proposed a symmetric version of the authenticated proxy re-encryption scheme [181]. The other results are ciphertext-policy attribute-based proxy re-encryption that can be used to restrict data sharing in the cloud [182–185].

7.3 Suggestions for the Future Work

The future work in the provenance research is to investigate application of our approach to a wider range of fields. The applications and security models of provenance can be different depend on the needs of the organizations. We may need to adapt the provenance system and the security model to the specific applications/organizations. We also may need to adapt or combine our method with the block-chain technology.

In the proxy re-encryption research, the future work is to apply the scheme in real cloud system. An interesting question is how to solve the problem of collusion between the proxy and the users. Is it possible to prevent the collusion in the symmetric encryption setting? The other interesting future work is to find other applications of All-or-Nothing Transform (AONT).

Appendix A

Implementation and Experimental Results

In this chapter, we describe our implementation and experimental results to the scheme described in Chapter 3 and part of the scheme in Chapter 4. In our implementation, we develop three applications that represent three parties as follows:

1. The Process Executor, that is the actor that creates provenance assertions $PAsrt$, creates the signature on $PAsrt$, and sends the assertion and the signature to the Provenance Store Interface (PSI).

2. The Provenance Store Interface (PSI) is the party that creates the secure provenance assertions $SPAsrt$. The PSI receives the provenance and its signature from the Process Executor, checks the signature, and requests the counter to the The Trusted Counter Server (TCS). The Provenance Store Interface also encrypts the provenance assertions and stores the provenance assertion to the provenance store.

3. The Trusted Counter Server (TCS), that is the trusted entity that receives requests for signed counters from the Provenance Store Interface (PSI), records the request and replies with the number of requests that have been received from the Provenance Store Interface (PSI).

A.1 Experimental Setup

We implemented the process executor with the Java SE 6 (JCE library for the cryptographic functions: SHA1 for hash and DSA for signature). In our experiments, the Process Executor sends the provenance assertions to the Provenance Store Interface using the HTTP Post protocol. The Provenance Store Interface is implemented with a PHP program that run in an Apache Web Server. The Provenance Store is implemented with a Postgresql database.

The Provenance Store Interface (PSI) also uses HTTP Post method to send the requests to the Trusted Counter Server (TCS). The TCS is also implemented with a PHP and an Apache web server. The TCS uses a Postgresql database for storing the counter. The cryptographic functions for the digital signature (DSA) in the Provenance Store Interface and the TCS are implemented the using OpenSSL library while the SHA1 supported natively by PHP. We also use an AES implementation for Windows for encryption[1].

We performed experiments in four computers where the first computer acts as a Process Executor, the second computer acts as the Provenance Store Interface, the third computer is the Trusted Counter Server (TCS) and the fourth computer is the Provenance Store. All of them are connected by a LAN with speed 100MBps. In Table A.1 below, we show the detail specification of the hardware and software we used in the experiments.

TABLE A.1: Hardware and Software of experiment

Role	Hardware	Software
Process Executor	Dual-Core 2.50GHz, 3GB RAM	Java SE 6 (JCE lib.), Windows XP
Provenance Store Interface	Dual-Core 2.50GHz, 3GB RAM	PHP 5.3.6, OpenSSL lib. ver. 0.9.8, Apache Web Server ver. 2.2.17, Postgresql ver. 9.0.4, Windows XP
Provenance Store	Core 2 Duo 1.4GHz, 4GB RAM	Postgresql 8.4.8, Linux 2.6.32

We performed 26 experiments to measure the performance of the scheme. For each experiment, we executed the process executors that send the provenance assertions to the Provenance Store Interface (PSI). After receiving each provenance

[1] http://www.aescrypt.com/

assertion, the Provenance Store Interface (PSI) requests the counters and stores the assertion to the Provenance Store. In our experiments, we simulated execution of 26 process executors where each process executor submits the provenance assertion with the range of size from 10KB to 1237KB (each provenance assertion is a text file that describes the execution of a process, the inputs, its outputs and the executing agents). We executed each experiment 12 (twelve) times and measured the execution times of various tasks that are needed to submit the provenance to the Provenance Store. Those tasks are as follows:

1. Hash-sign: the execution time that is needed to create hash and signature of the process documentation by the Process Executor.
2. Upload: the time to upload the signed provenance assertion and its signature to the Provenance Store Interface.
3. Encrypt1: the execution time that is needed to generate the node key (KN) and to encrypt the provenance assertion with the node key.
4. Encrypt2: the execution time that is needed to encrypt the provenance assertion with the label key (KL).
5. Req-Counter: the execution time that is needed by the PSI to send the request to the TCS and to receive the response from the TCS.
6. Counter: the execution time that is needed to calculate the counter by the TCS.
7. Store: the execution time that is needed to upload the data to the Provenance Store.

A.2 Results

Figures A.1, A.2, and A.3 show the average of the execution times of the tasks. The X axis is the size of the process documentation (provenance assertion) in kilobytes (KB), the Y axis is the execution time (in seconds) for each process executor, Provenance Store Interface and the TCS. We summarize the complexity of the the execution of each task relative to the size of the process documentation A in Table A.2.

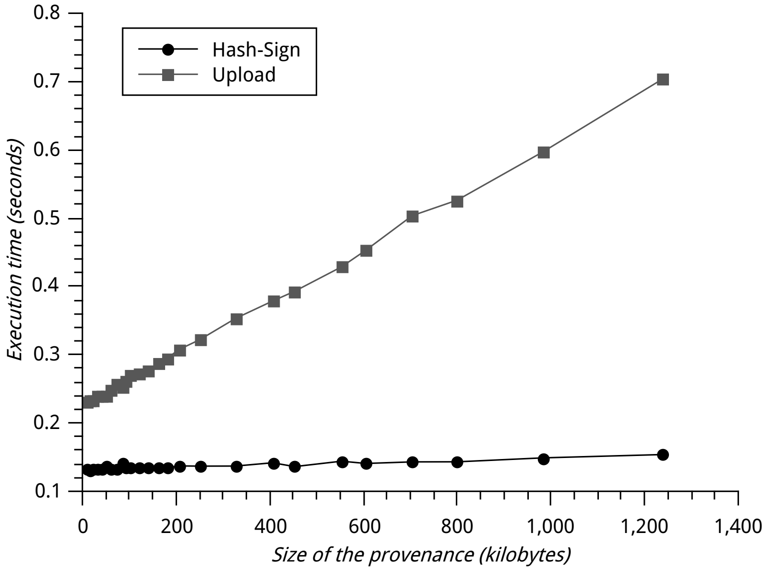

FIGURE A.1: Execution time of the Provenance Executor (in seconds)

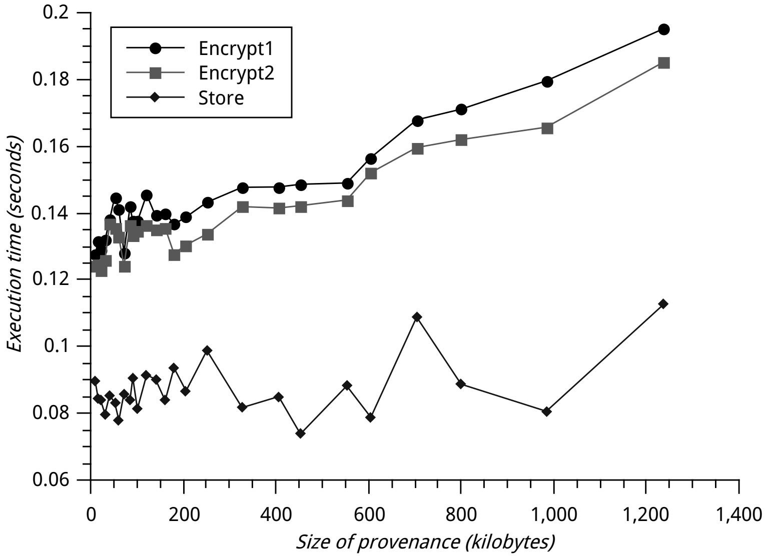

FIGURE A.2: Execution time of the Provenance Store Interface (in seconds)

As described in Table A.2, for the process executor, the time to create the signature is almost constant. This result shows that there is not much difference in the execution time needed to create signature and hash of files in the range of size of the process documentation A (10KB to 1237KB) and constant size of outputs. The time to upload the provenance assertion to the Provenance Store Interface is linear to the size of the process documentation. This result is natural because the time needed to send the provenance assertion via the network is linear with the size of the data.

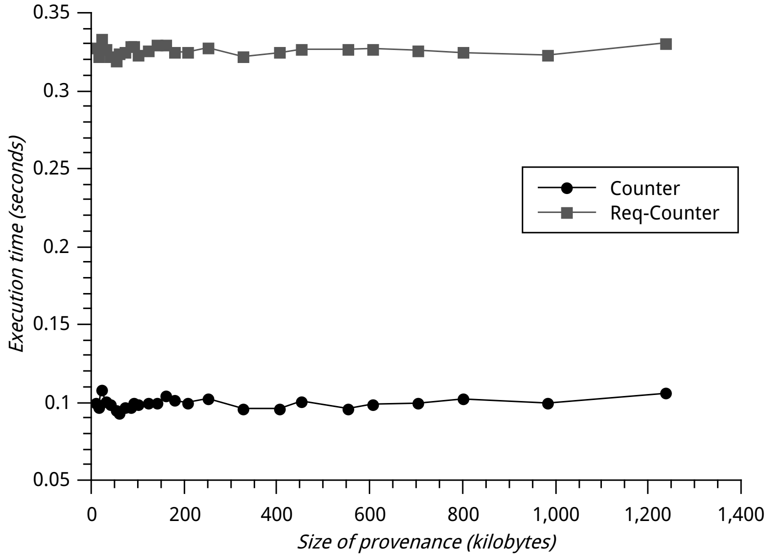

FIGURE A.3: Execution time of the TCS (in seconds)

TABLE A.2: The complexity of each task (relative to the size of process documentation A)

Role	Task	Complexity
Process Executor	Hash-Sign Upload	Almost constant Linear
Provenance Store Interface	Check-PrepReq Encrypt1 Encrypt2 Store	Almost Constant Linear (with small growth) Linear (with small growth) Almost constant
TCS	Counter ReqCounter	Constant Constant

As of the Provenance Store Interface, the time to encrypt and to store the provenance assertions is slightly increased with size. An interesting result is the growth of execution time to submit the provenance assertion to the provenance store is almost constant, and this growth is different from the growth of time needed by the process executor to upload the provenance assertion to the Provenance Store Interface. These results show that the time that is needed to upload data using HTTP Post protocol and to store the data to a filesystem that are used by the process executor to send the provenance assertion to the Provenance Store Interface is much slower than the protocol to store the data to a Postgresql database using Postgresql library used by the Provenance Store Interface to submit the provenance assertion to the Provenance Store.

The time that is needed by the TCS to compute the counter and the total time to send request and to receive the counter to/from the TCS are also constant. These results are natural because the times to check the signature, to increase counter and to prepare the hash are constant. The time to prepare the hash is also constant because the size of the requests is constant (the request consists of the provenance's id, the hash of the provenance assertion and a timestamp). Because the size of the requests to the TCS and reply (counter) from the TCS are constant, the time that is needed to send the request and to receive the response using the network is also constant.

Our experimental results show the feasibility to implement our scheme in a real system, because most of the execution times that are needed in the scheme (except for the time needed to upload the provenance assertion to the provenance store) are almost constant or with the small growth. Even in our hardware configuration (which is a basic configuration) the time for the TCS to create the counter is around 0.1 seconds and the total time including the network costs (receiving the request and replying with the counter) is not more than 0.5 seconds while for the Provenance Store Interface the total time for all tasks except requesting the counter is not more than 0.6 seconds. As for the time to upload the provenance assertion to the Provenance Store Interface, our results suggest the usage of a better or a faster communication protocol and storage than the HTTPS protocol and the normal filesystem.

Published Papers

Journal Papers

1. Amril Syalim, Takashi Nishide, Kouichi Sakurai. Securing Provenance of Distributed Processes in an Untrusted Environment. *IEICE Transactions on Information and Systems*, Vol. E95-D, No.7, pp.1894-1907, 2012.

2. Amril Syalim, Toshihiro Tabata and Kouichi Sakurai. Usage Control Model and Architecture for Data Confidentiality in a Database Service Provider. *IPSJ Journal* (Technical Note), Vol.47, No.2, pp.621-626, 2006.

International Conference Papers

1. Amril Syalim, Takashi Nishide, Kouichi Sakurai. Improved Proxy Re-encryption Scheme for Symmetric Key Cryptography. *Proceedings of International Workshop on Big Data and Information Security (IWBIS) 2017*, IEEE, pp. 105 - 111, 2017. (Best Paper Award)

2. Amril Syalim, Kouichi Sakurai. How to Sign Multiple Versions of Digital Documents. *Proceedings of International Workshop on Big Data and Information Security (IWBIS) 2017*, IEEE, pp. 133 - 136, 2017.

3. Amril Syalim, Takashi Nishide, Kouichi Sakurai. Supporting Secure Provenance Update by Keeping "Provenance" of the Provenance. *Proceedings of the ICT-EurAsia 2013*, Springer Verlag, LNCS 7804, pp. 363-372, 2013.

4. Amril Syalim, Takashi Nishide, Kouichi Sakurai. Realizing Proxy Re-encryption in the Symmetric World. *Proceedings of the International Conference on Informatics Engineering and Information Science (ICIEIS2011)*, Springer Verlag, Communication in Computer and Information Science 251, pp. 259-274, 2011.

5. Amril Syalim, Takashi Nishide, Kouichi Sakurai. Preserving Integrity and Confidentiality of a Directed Acyclic Graph of Provenance. *Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2010)*, Springer Verlag, Lecture Notes in Computer Science 6166, pp. 311-318, 2010.

6. Amril Syalim, Yoshiaki Hori, Kouichi Sakurai. Grouping Provenance Information to Improve Efficiency of Access Control. *Proceeding of the Third International Conference and Workshops on Advances in Information Security and Assurance (ISA 2009)*, Springer Verlag, Lecture Notes in Computer Science 5576, pp. 51-59, 2009.

7. Amril Syalim, Yoshiaki Hori, Kouichi Sakurai. Comparison of Risk Analysis Methods: Mehari, Magerit, NIST800-30 and Microsoft's Security Management Guide. *The First International Workshop on Organizational Security Aspects (OSA 2009)*, pp.726-731, 2009.

8. Amril Syalim, Toshihiro Tabata and Kouichi Sakurai. Usage Control Model and Architecture for Data Confidentiality in Database Service Provider. *Indonesia Cryptology and Information Security Conference (INA-CISC)* 2005, pp. 155-160, 2005.

Bibliography

[1] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani M. Thuraisingham. A language for provenance access control. In *CODASPY*, pages 133–144, 2011.

[2] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An architecture for provenance systems. Technical report, University of Southampton, November 2006.

[3] Vikas Deora, Arnaud Contes, Omer F. Rana, Shrija Rajbhandari, Ian Wootten, Tamás Kifor, and László Zsolt Varga. Navigating provenance information for distributed healthcare management. In *Web Intelligence*, pages 859–865, 2006.

[4] Kiran-Kumar Muniswamy-Reddy. *Foundations for Provenance-Aware Systems.* PhD thesis, Harvard University, Cambridge, Massachusetts, March 2010.

[5] Ragib Hasan, Radu Sion, and Marianne Winslett. Preventing history forgery with secure provenance. *ACM Transactions on Storage*, 5(4):12:1–12:43, December 2009.

[6] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *In EUROCRYPT*, pages 127–144. Springer-Verlag, 1998.

[7] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

[8] R. Canetti and S. Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 185–194. ACM, 2007.

[9] M. Green and G. Ateniese. Identity-based proxy re-encryption. In *Applied Cryptography and Network Security*, pages 288–306. Springer, 2007.

[10] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Transactions on Information Theory*, 57(3): 1786–1802, 2011.

[11] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[12] Maarten van Steen and Andrew S Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, 2016.

[13] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*, pages 44–51. Ieee, 2009.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53 (4):50–58, 2010.

[15] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[16] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[17] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[18] Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. In *SIGMOD Record*, pages 31–36, 2005.

[19] Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of e-science experiment provenance. *Concurrency and Computation: Practice and Experience*, 20(5):419–429, 2008.

[20] Simon Miles, Paul T. Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *J. Grid Comput.*, 5(1): 1–25, 2007.

[21] Simon Miles, Sylvia C. Wong, Weijian Fang, Paul T. Groth, Klaus-Peter Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. *J. Web Sem.*, 5(1):28–38, 2007.

[22] Tamás Kifor, László Zsolt Varga, Javier Vázquez-Salceda, Sergio Álvarez-Napagao, Steven Willmott, Simon Miles, and Luc Moreau. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems*, 21 (6):38–46, 2006.

[23] Sergio Álvarez-Napagao, Javier Vázquez-Salceda, Tamás Kifor, László Zsolt Varga, and Steven Willmott. Applying provenance in distributed organ transplant management. In *IPAW*, pages 28–36, 2006.

[24] Min Wang, Marion Blount, John Davis, Archan Misra, and Daby M. Sow. A time-and-value centric provenance model and architecture for medical event streams. In *HealthNet*, pages 95–100, 2007.

[25] Ueli M. Maurer. New approaches to digital evidence. *Proceedings of the IEEE*, 92(6):933–947, 2004.

[26] Martin Schäler, Sandro Schulze, and Stefan Kiltz. Database-centric chain-of-custody in biometric forensic systems. In *BIOID*, pages 250–261, 2011.

[27] J A Simpson and E S C Weiner. *The Oxford English dictionary.* Oxford University Press, 1989.

[28] Luc Moreau, Paul T. Groth, Simon Miles, Javier Vázquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer F. Rana, Andreas Schreiber, Victor Tan, and László Zsolt Varga. The provenance of electronic data. *Communications of the ACM*, 51(4):52–58, 2008.

[29] Wang Chiew Tan. Research problems in data provenance. *IEEE Data Eng. Bull.*, 27(4):45–52, 2004.

[30] Wang Chiew Tan. Provenance in databases: Past, current, and future. *IEEE Data Eng. Bull.*, 30(4):3–12, 2007.

[31] Luc Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2-3):99–241, 2010.

[32] Paul T. Groth and Luc Moreau. Recording process documentation for provenance. *IEEE Trans. Parallel Distrib. Syst.*, 20(9):1246–1259, 2009.

[33] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul T. Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric G. Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Comp. Syst.*, 27(6):743–756, 2011.

[34] Boris Glavic and Klaus R. Dittrich. Data provenance: A categorization of existing approaches. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007)*, pages 227–241, 2007.

[35] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *International Conference on Database Theory (ICDT 2001), LNCS 1973*, pages 316–330, 2001.

[36] Simon Miles, Paul T. Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. Extracting causal graphs from an open provenance data model. *Concurrency and Computation: Practice and Experience*, 20 (5):577–586, 2008.

[37] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 367–378. IEEE, 2000.

[38] Peter Buneman, Adriane Chapman, James Cheney, and Stijn Vansummeren. A provenance model for manually curated data. In *IPAW*, pages 162–170, 2006.

[39] Bertram Ludascher, Norbert Podhorszki, Ilkay Altintas, Shawn Bowers, and Timothy M. McPhillips. From computation models to models of provenance: the rws approach. *Concurrency and Computation: Practice and Experience*, 20(5):507–518, 2008.

[40] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *ACM workshop on Storage security and survivability (StorageSS 2007)*, pages 13–18, 2007.

[41] Ragib Hasan, Radu Sion, and Marianne Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *7th Conference on File and Storage Technologies (FAST 2009)*, pages 1–14, 2009.

[42] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A framework for collecting provenance in data-centric scientific workflows. In *IEEE International Conference on Web Services (ICWS 2006)*, pages 427–436, 2006.

[43] Shawn Bowers, Timothy M. McPhillips, Bertram Ludascher, Shirley Cohen, and Susan B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *International Provenance and Annotation Workshop (IPAW 2006), LNCS 4145*, pages 133–147, 2006.

[44] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security (HOTSEC 2008)*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, July 2008. USENIX Association.

[45] Shirley Cohen, Sarah Cohen-Boulakia, and Susan Davidson. Towards a model of provenance and user views in scientific workflows. In *Data Integration in the Life Sciences*, pages 264–279. Springer, 2006.

[46] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21, 2008.

[47] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. The open provenance model. Technical report, University of Southampton, 2007.

[48] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The open provenance model: An overview. In *IPAW*, pages 323–326, 2008.

[49] Marc J Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, (4):364–370, 1975.

[50] Alan L Glasser. The evolution of a source code control system. *ACM SIGSOFT Software Engineering Notes*, 3(5):122–125, 1978.

[51] Walter F Tichy. Rcsa system for version control. *Software: Practice and Experience*, 15(7):637–654, 1985.

[52] Bryan O'Sullivan. Making sense of revision-control systems. *Communications of the ACM*, 52(9):56–62, 2009.

[53] Ben Collins-Sussman, Brian Fitzpatrick, and Michael Pilato. *Version control with subversion.* " O'Reilly Media, Inc.", 2004.

[54] Tom De Nies, Sara Magliacane, Ruben Verborgh, Sam Coppens, Paul T Groth, Erik Mannens, and Rik Van de Walle. Git2prov: Exposing version control system content as w3c prov. In *International Semantic Web Conference (Posters & Demos)*, pages 125–128, 2013.

[55] David Koop, Emanuele Santos, Bela Bauer, Matthias Troyer, Juliana Freire, and Cláudio T Silva. Bridging workflow and data provenance using strong links. In *International Conference on Scientific and Statistical Database Management*, pages 397–415. Springer, 2010.

[56] James Cheney, Stephen Chong, Nate Foster, Margo Seltzer, and Stijn Vansummeren. Provenance: a future history. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 957–964. ACM, 2009.

[57] Harry Halpin and James Cheney. Dynamic provenance for sparql updates. In *International Semantic Web Conference*, pages 425–440. Springer, 2014.

[58] T. Kifor, L.Z. Varga, S. Álvarez, J. Vázquez-Salceda, and S. Willmott. Privacy issues of provenance in electronic healthcare record systems. In *Proceedings of the 1st Int. Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE 2006)*, 2006.

[59] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track*, pages 43–56, 2006.

[60] Robert Ikeda and Jennifer Widom. Panda: A system for provenance and data. *IEEE Data Eng. Bull.*, 33(3):42–49, 2010.

[61] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms.* Prentice-Hall, 2007.

[62] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49, 2005.

[63] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, 3(3-4):171–200, 2005.

[64] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *International Workshop on Peer-to-Peer Systems*, pages 205–216. Springer, 2005.

[65] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[66] Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In *UK e-Science All Hands Meeting (AHM 2005)*, September 2005.

[67] Paul T. Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In *OPODIS*, pages 124–139, 2004.

[68] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[69] Oded Goldreich. Foundation of cryptography (in two volumes: Basic tools and basic applications). 2001.

[70] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.

[71] Ralf Senderek. A discrete logarithm hash function for rsa signatures, 2003. URL http://www.senderek.ie/research/pcp/release/doc/discrete-logarithm-hash-for-RSA-signatures.ps.

[72] Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding sha-2 characteristics: searching through a minefield of contradictions. In *Advances in Cryptology–ASIACRYPT 2011*, pages 288–307. Springer, 2011.

[73] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for step-reduced sha-2. In *Advances in Cryptology–ASIACRYPT 2009*, pages 578–597. Springer, 2009.

[74] Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step sha-2. In *Progress in Cryptology-INDOCRYPT 2008*, pages 91–103. Springer, 2008.

[75] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[76] Matt Bishop. *Computer security: art and science.* Addison-Wesley Professional, 2003.

[77] Stallings William. *Cryptography and Network Security: Principles and Practice (6th Edition).* Pearson, 2013.

[78] Data Encryption Standard et al. Federal information processing standards publication 46. *National Bureau of Standards, US Department of Commerce*, 4, 1977.

[79] NIST-FIPS Standard. Announcing the advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:1–51, 2001.

[80] Lars R Knudsen. Practically secure feistel ciphers. In *International Workshop on Fast Software Encryption*, pages 211–221. Springer, 1993.

[81] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[82] M. Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, National Institute of Standards and Technology, 2001.

[83] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

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

[85] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[86] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 13–25. Springer, 1998.

[87] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[88] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[89] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[90] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1984.

[91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[92] P Gallagher and C Kerry. Fips pub 186-4: Digital signature standard (dss), 2013.

[93] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Advances in CryptologyASIACRYPT 2001*, pages 514–532, 2001.

[94] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP*, pages 411–422, 2007.

[95] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *TOS*, 2(2):107–138, 2006.

[96] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[97] Mihir Bellare. Practice-oriented provable-security. In *International Workshop on Information Security*, pages 221–231. Springer, 1997.

[98] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM*

conference on Computer and communications security, pages 62–73. ACM, 1993.

[99] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[100] Victor Shoup. Oaep reconsidered. *Journal of Cryptology*, 15(4):223–249, 2002.

[101] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[102] Jean-Sébastien Coron. On the exact security of full domain hash. In *Annual International Cryptology Conference*, pages 229–235. Springer, 2000.

[103] Rocío Aldeco-Pérez and Luc Moreau. Securing provenance-based audits. In *IPAW*, pages 148–164, 2010.

[104] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Preserving integrity and confidentiality of a directed acyclic graph model of provenance. In *IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2010), LNCS 6166*, pages 311–318, 2010.

[105] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991.

[106] Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In *ASIACRYPT*, pages 500–514, 2004.

[107] Ahto Buldas and Margus Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In *ACISP*, pages 318–335, 2010.

[108] Luiz M. R. Gadelha and Marta Mattoso. Kairos: An architecture for securing authorship and temporal information of provenance data in grid-enabled workflow management systems. In *2008 IEEE Fourth International Conference on eScience*, pages 597–602. IEEE, dec 2008. ISBN 978-1-4244-3380-3. doi: 10.1109/eScience.2008.161.

[109] Paul T Groth. *The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes.* PhD thesis, University of Southampton, 2007.

[110] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[111] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[112] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[113] Uri Braun and Avi Shinnar. A security model for provenance. Technical report, Harvard University, 2006.

[114] Victor Tan, Paul T. Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security issues in a soa-based provenance system. In *International Provenance and Annotation Workshop (IPAW 2006), LNCS 4145*, pages 203–211, 2006.

[115] Artem Chebotko, Seunghan Chang, Shiyong Lu, Farshad Fotouhi, and Ping Yang. Scientific workflow provenance querying with security views. In *WAIM*, pages 349–356. IEEE press, 2008.

[116] Meiyappan Nagappan and Mladen A. Vouk. A model for sharing of confidential provenance information in a query based system. In *International Provenance and Annotation Workshop (IPAW 2008), LNCS 5272*, June 2008.

[117] Qun Ni, Shouhuai Xu, Elisa Bertino, Ravi S. Sandhu, and Weili Han. An access control language for a general provenance model. In *Secure Data Management*, pages 68–88, 2009.

[118] Security Frameworks for Open Systems: Access Control Framework. Technical Report Recommendation X.812, ITU-T, 1995.

[119] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Securing provenance of distributed processes in an untrusted environment. *IEICE TRANSACTIONS on Information and Systems*, 95(7):1894–1907, 2012.

[120] Yumin Yuan, Qian Zhan, and Hua Huang. Efficient unrestricted identity-based aggregate signature scheme. *PloS one*, 9(10):e110100, 2014.

[121] Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology (EUROCRYPT 98)*, pages 236–250. Springer, 1998.

[122] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 74–90. Springer, 2004.

[123] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–485. Springer, 2006.

[124] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 276–285. ACM, 2007.

[125] Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 86–91. IEEE, 2007.

[126] Yi Mu, Willy Susilo, and Huafei Zhu. Compact sequential aggregate signatures. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 249–253. ACM, 2007.

[127] Gregory Neven. Efficient sequential aggregate signed data. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 52–69. Springer, 2008.

[128] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations. *Information and computation*, 239:356–376, 2014.

[129] Jia-Lun Tsai, Nai-Wei Lo, and Tzong-Chen Wu. New identity-based sequential aggregate signature scheme from rsa. In *Biometrics and Security Technologies (ISBAST), 2013 International Symposium on*, pages 136–140. IEEE, 2013.

[130] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures with short public keys without random oracles. *Theoretical Computer Science*, 579:100–125, 2015.

[131] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures made shorter. In *International Conference on Applied Cryptography and Network Security*, pages 202–217. Springer, 2013.

[132] Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. In *International Conference on Security and Cryptography for Networks*, pages 113–130. Springer, 2012.

[133] Oliver Eikemeier, Marc Fischlin, Jens-Fabian Götzmann, Anja Lehmann, Dominique Schröder, Peter Schröder, and Daniel Wagner. History-free aggregate message authentication codes. In *International Conference on Security and Cryptography for Networks*, pages 309–328. Springer, 2010.

[134] Craig Gentry, Adam ONeill, and Leonid Reyzin. A unified framework for trapdoor-permutation-based sequential aggregate signatures. In *IACR International Workshop on Public Key Cryptography*, 2018.

[135] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the rsa assumption. 2018.

[136] Kaisa Nyberg and Rainer A Rueppel. A new signature scheme based on the dsa giving message recovery. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 58–61. ACM, 1993.

[137] Kaisa Nyberg and Rainer A Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 182–193. Springer, 1994.

[138] Kaisa Nyberg and Rainer A Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7(1-2):61–81, 1996.

[139] Atsuko Miyaji. A message recovery signature scheme equivalent to dsa over elliptic curves. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–14. Springer, 1996.

[140] Masayuki Abe and Tatsuaki Okamoto. A signature scheme with message recovery as secure as discrete logarithm. *Advances in Cryptology-ASIACRYPT'99*, pages 378–389, 1999.

[141] Debra L. Cook and Angelos D. Keromytis. Conversion and proxy functions for symmetric key ciphers. In *ITCC*, pages 662–667, 2005.

[142] Burton S. Kaliski Jr., Ronald L. Rivest, and Alan T. Sherman. Is the data encryption standard a group? (results of cycling experiments on des). *J. Cryptology*, 1(1):3–36, 1988.

[143] Shoichi Hirose. On re-encryption for symmetric authenticated encryption. In *Computer Security Symposium (CSS) 2010*, 2010.

[144] Ronald L. Rivest. All-or-nothing encryption and the package transform. In *In Fast Software Encryption, LNCS 1267*, pages 210–218. Springer-Verlag, 1997.

[145] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.

[146] Victor Boyko. On the security properties of oaep as an all-or-nothing transform. In *CRYPTO*, pages 503–518, 1999.

[147] Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453–469, 2000.

[148] Yevgeniy Dodis, Amit Sahai, and Adam Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT*, pages 301–324, 2001.

[149] Anand Desai. The security of all-or-nothing encryption: Protecting against exhaustive key search. In *CRYPTO*, pages 359–375, 2000.

[150] Douglas R. Stinson. Something about all or nothing (transforms). *Des. Codes Cryptography*, 22(2):133–138, 2001.

[151] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO (1)*, pages 410–428, 2013.

[152] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Realizing proxy re-encryption in the symmetric world. In *International Conference on Informatics Engineering and Information Science*, pages 259–274. Springer, 2011.

[153] Mihir Bellare and Phil Rogaway. The game-playing technique. *International Association for Cryptographic Research (IACR) ePrint Archive: Report*, 331: 2004, 2004.

[154] Mohammad M Bany Taha, Sivadon Chaisiri, and Ryan KL Ko. Trusted tamper-evident data provenance. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 646–653. IEEE, 2015.

[155] Liang Chen, Peter Edwards, John D Nelson, and Timothy J Norman. An access control model for protecting provenance graphs. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, pages 125–132. IEEE, 2015.

[156] Yulai Xie, Dan Feng, Zhipeng Tan, and Junzhe Zhou. Unifying intrusion detection and forensic analysis via provenance awareness. *Future Generation Computer Systems*, 61:26–36, 2016.

[157] Jiun Yi Yap and Allan Tomlinson. Provenance-based attestation for trustworthy computing. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 630–637. IEEE, 2015.

[158] Jun Chen, Tun Lu, Guo Li, Tiejiang Liu, Xianghua Ding, and Ning Gu. Provenance based diagnosis for scientific workflows. In *Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on*, pages 549–554. IEEE, 2015.

[159] Richard McClatchey, Jetendr Shamdasani, Andrew Branson, Kamran Munir, Zsolt Kovacs, and Giovanni Frisoni. Traceability and provenance in big data medical systems. In *Computer-Based Medical Systems (CBMS), 2015 IEEE 28th International Symposium on*, pages 226–231. IEEE, 2015.

[160] Fuzel Jamil, Abid Khan, Adeel Anjum, Mansoor Ahmed, Farhana Jabeen, and Nadeem Javaid. Secure provenance using an authenticated data structure approach. *Computers & Security*, 73:34–56, 2018.

[161] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE, 2016.

[162] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[163] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[164] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[165] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.

[166] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. In *International Conference on Business Process Management*, pages 329–347. Springer, 2016.

[167] Jan Mendling, Ingo Weber, Wil Van Der Aalst, Jan Vom Brocke, Cristina Cabanillas, Florian Daniel, Søren Debois, Claudio Di Ciccio, Marlon Dumas, Schahram Dustdar, et al. Blockchains for business process management-challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 9(1):4, 2018.

[168] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. 2016.

[169] Haoyan Wu, Zhijie Li, Brian King, Zina Ben Miled, John Wassick, and Jeffrey Tazelaar. A distributed ledger for supply chain physical distribution visibility. *Information*, 8(4):137, 2017.

[170] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In

Open and Big Data (OBD), International Conference on, pages 25–30. IEEE, 2016.

[171] Ariel Ekblaw, Asaph Azaria, John D Halamka, and Andrew Lippman. A case study for blockchain in healthcare:medrec prototype for electronic health records and medical research data. In *Proceedings of IEEE Open & Big Data Conference*, volume 13, page 13, 2016.

[172] Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang. Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control. *Journal of medical systems*, 40(10):218, 2016.

[173] Laure A Linn and Martha B Koo. Blockchain for health data and its potential use in health it and health care related research. In *ONC/NIST Use of Blockchain for Healthcare and Research Workshop, Gaithersburg, Maryland, United States: ONC/NIST*, 2016.

[174] Mike Sharples and John Domingue. The blockchain and kudos: A distributed system for educational record, reputation and reward. In *European Conference on Technology Enhanced Learning*, pages 490–496. Springer, 2016.

[175] Janusz J Sikorski, Joy Haughton, and Markus Kraft. Blockchain technology in the chemical industry: Machine-to-machine electricity market. *Applied Energy*, 195:234–246, 2017.

[176] Yang Lu and Jiguo Li. A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds. *Future Generation Computer Systems*, 2015.

[177] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography*, pages 53–68. Springer, 1999.

[178] S Canard and J Devigne. Highly privacy-protecting data sharing in a tree structure. *Future Generation Computer Systems*, 2016.

[179] Jun Shao, Rongxing Lu, Xiaodong Lin, and Kaitai Liang. Secure bidirectional proxy re-encryption for cryptographic cloud storage. *Pervasive and Mobile Computing*, 2015.

[180] Satsuya Ohata, Yutaka Kawai, Takahiro Matsuda, Goichiro Hanaoka, and Kanta Matsuura. Re-encryption verifiability: How to detect malicious activities of a proxy in proxy re-encryption. In *Cryptographers Track at the RSA Conference*, pages 410–428. Springer, 2015.

[181] Zhiniang Peng, Shaohua Tang, and Linzhi Jiang. A symmetric authenticated proxy re-encryption scheme with provable security. In *International Conference on Cloud Computing and Security*, pages 86–99. Springer, 2017.

[182] Qin Liu, Guojun Wang, and Jie Wu. Time-based proxy re-encryption scheme for secure data sharing in a cloud environment. *Information Sciences*, 258: 355–370, 2014.

[183] Kaitai Liang, Man Ho Au, Joseph K Liu, Willy Susilo, Duncan S Wong, Guomin Yang, Yong Yu, and Anjia Yang. A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing. *Future Generation Computer Systems*, 52:95–108, 2015.

[184] Yanjiang Yang, Haiyan Zhu, Haibing Lu, Jian Weng, Youcheng Zhang, and Kim-Kwang Raymond Choo. Cloud based data sharing with fine-grained proxy re-encryption. *Pervasive and Mobile computing*, 28:122–134, 2016.

[185] David Nuez, Isaac Agudo, and Javier Lopez. Proxy re-encryption. *Journal of Network and Computer Applications*, 87(C):193–209, 2017.

Index