Secure Schemes for Semi-Trusted Environment

by

Anuchart Tassanaviboon

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In recent years, two distributed system technologies have emerged: Peer-to-Peer (P2P) and cloud computing. For the former, the computers at the edge of networks share their resources, i.e., computing power, data, and network bandwidth, and obtain resources from other peers in the same community. Although this technology enables efficiency, scalability, and availability at low cost of ownership and maintenance, peers defined as "like each other" are not wholly controlled by one another or by the same authority. In addition, resources and functionality in P2P systems depend on peer contribution, i.e., storing, computing, routing, etc. These specific aspects raise security concerns and attacks that many researchers try to address. Most solutions proposed by researchers rely on public-key certificates from an external Certificate Authority (CA) or a centralized Public Key Infrastructure (PKI). However, both CA and PKI are contradictory to fully decentralized P2P systems that are self-organizing and infrastructureless.

To avoid this contradiction, this thesis concerns the provisioning of public-key certificates in P2P communities, which is a crucial foundation for securing P2P functionalities and applications. We create a framework, named the **Self-Organizing and Self-Healing CA group (SOHCG)**, that can provide certificates without a centralized Trusted Third Party (TTP). In our framework, a CA group is initialized in a Content Addressable Network (CAN) by trusted bootstrap nodes and then grows to a mature state by itself. Based on our group management policies and predefined parameters, the membership in a CA group is dynamic and has a uniform distribution over the P2P community; the size of a CA group is kept to a level that balances performance and acceptable security. The muticast group over an underlying CA group is constructed to reduce communication and computation overhead from collaboration among CA members. To maintain the quality of the CA group, the honest majority of members is maintained by a Byzantine agreement algorithm, and all shares are refreshed gradually and continuously. Our CA framework has been designed to meet all design goals, being self-organizing, self-healing, scalable, resilient, and efficient. A security analysis shows that the framework enables key registration and certificate issue with resistance to external attacks, i.e., node impersonation, man-in-the-middle (MITM), Sybil, and a specific form of DoS, as well as internal attacks, i.e., CA functionality interference and CA group subversion.

Cloud computing is the most recent evolution of distributed systems that enable shared resources like P2P systems. Unlike P2P systems, cloud entities are asymmetric in roles like client-server models, i.e., end-users collaborate with Cloud Service Providers (CSPs) through Web interfaces or Web portals. Cloud computing is a combination of technologies, e.g., SOA services, virtualization, grid computing, clustering, P2P overlay networks,

management automation, and the Internet, etc. With these technologies, cloud computing can deliver services with specific properties: on-demand self-service, broad network access, resource pooling, rapid elasticity, measured services. However, theses core technologies have their own intrinsic vulnerabilities, so they induce specific attacks to cloud computing. Furthermore, since public clouds are a form of outsourcing, the security of users' resources must rely on CSPs' administration. This situation raises two crucial security concerns for users: locking data into a single CSP and losing control of resources. Providing interoperations between Application Service Providers (ASPs) and untrusted cloud storage is a countermeasure that can protect users from lock-in with a vendor and losing control of their data.

To meet the above challenge, this thesis proposed a new authorization scheme, named **OAuth and ABE based authorization (AAuth)**, that is built on the OAuth standard and leverages Ciphertext-Policy Attribute Based Encryption (CP-ABE) and ElGamal-like masks to construct ABE-based tokens. The ABE-tokens can facilitate a user-centric approach, end-to-end encryption and end-to-end authorization in semi-trusted clouds. With these facilities, owners can take control of their data resting in semi-untrusted clouds and safely use services from unknown ASPs. To this end, our scheme divides the attribute universe into two disjointed sets: confined attributes defined by owners to limit the lifetime and scope of tokens and descriptive attributes defined by authority(s) to certify the characteristic of ASPs. Security analysis shows that AAuth maintains the same security level as the original CP-ABE scheme and protects users from exposing their credentials to ASP, as OAuth does. Moreover, AAuth can resist both external and internal attacks, including untrusted cloud storage. Since most cryptographic functions are delegated from owners to CSPs, AAuth gains computing power from clouds. In our extensive simulation, AAuth's greater overhead was balanced by greater security than OAuth's. Furthermore, our scheme works seamlessly with storage providers by retaining the providers' APIs in the usual way.

Acknowledgements

First, I would like to thank my supervisor, Prof. Guang Gong, for her patience and support during my studies. She has taught me a lot, and I am grateful. Thanks are also due to my committee members Prof. Lein Harn at University of Missouri Kansas City as well as Prof. Grace Yi, Prof. Anwar M. Hasan, and Prof. Paul Ward at University of Waterloo for serving as my examiners and providing me with insights that have helped me improve my work.

My office mates deserve thanks as well; together we have weathered some very stressful times. In addition, Mary McPherson of the English Language Proficiency Program has helped me with my English. Above all, my wife Nok deserves my eternal gratitude and affection for putting up with the stress and chaos of the last few months and for believing in me. Thanks too to our adorable but noisy baby Maple, who has not helped my work in the least but who is nevertheless a joy and an inspiration.

I would like to thank all the "little people" who made this possible; that is all the many people who smooth our daily lives but whom we rarely know by name. University life runs on their efforts.

None of my work here would have been possible without the faith in my capabilities from Thai agencies: the faculties of King Mongkut's University Thonburi and the management team of the Ministry of Science and Technology. Sending me here and supporting me has given me a unique chance to gain skills I hope will return benefits to them.

Dedication

This is dedicated to the ones I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computing systems were originally initialized from a centralized system that encompassed a single- or multiple- processor(s) unit, system software, and applications to form a single-unit solution. Centralized systems usually impress users with their ease of management and security. However, they are poor in terms of reliability, scalability, and economy. These disadvantages are addressed by the emergence of the next generation of computing systems, many distributed systems, in which components are located in computer networks and coordinate their actions by passing messages. Although distributed systems impose communication overhead and security weakness, their advantages (i.e., reliability, scalability, and economy) compensate for the drawbacks. Currently, distributed systems can be divided into three general models: client-server, peer-to-peer, and cloud computing.

In client-server models, computing systems partition tasks or workloads between the providers of resources or services (called servers) and service requesters (called clients). Since a server must support all requests from clients, the capacity of the server is a bottleneck and must be dimensioned in such a way that it can support all requests from clients at any time. Moreover, a single server lacks the robustness of a redundant configuration. If a critical server fails, no client requests can be fulfilled. Similar to centralized systems, the main advantages of client-server systems are their ease of management and security, because most processes/information are dealt with/stored in the server side. Peer-to-peer and cloud computing are recent distributed systems that gain efficiency, reliability, scalability from extensively sharing resources. However, their technologies impose security weakness. Therefore, this thesis addresses the security problems in P2P systems and cloud computing.

1.1 Peer-to-Peer (P2P)

A Peer-to-Peer (P2P) system is a distributed system formed by an overlay network consisting of a collection of nodes and links between a node and its neighbors. The overlay network is built on top of other underlying networks, such as IP networks, wireless networks, or sensor networks. The links are created and maintained by middleware located between the underlying network and application. The middleware can place nodes, called peers, on an overlay network in two ways: structured and unstructured. In *structured P2P systems*, peer IDs can be generated from random functions or deterministic functions taking other persistent IDs, such as IP addresses, as input. Then these peer IDs are mapped tightly to the topology of structured overlay networks, e.g., CAN [103], Chord [122], Pastry [4], Viceroy [80], Kademlia [83], Tapestry [136], etc. Without structure from overlay networks, *unstructured P2P systems* directly use network addresses for peers. P2P systems go beyond client-server models by having symmetry in roles where a peer acts simultaneously as a client and a server. This symmetry allows peers to leverage resources from multiple sharing peers rather than a single server. In this way, P2P systems will scale up network bandwidth and the number of sharing peers as network population increases. Consequently, P2P can support resource sharing with fault-tolerance, load balance, self-organization, and massive scalability. In addition to P2P construction (overlay networks and symmetric roles), P2P systems must have four main self-organizing functions: peer discovery, enrollment, resource indexing, and message routing, which are explained as follows.

(i) Peer discovery. Before joining a P2P system (i.e., becoming a peer), a node must make a connection with one or more peers already present in the overlay network. Thus, P2P systems must provide processes for new nodes to discover existing peers in the network.

(ii) Enrollment. This is a registration process used when peers join or leave the overlay network in order to initialize and update contract information, i.e., neighbor lists (links), resource indicies, and routing tables. In some systems, new nodes must obtain valid credentials before joining.

(iii) Resource indexing. Resource indices are the locations of resources in an overlay network. There are two main ways to place resources in the network:

1 Randomly selecting live peers from the network;

2 Using a cryptographic hash function to map resource keys to live peers.

The P2P applications on top of overlay networks will mainly use this indexing for resource lookups.

(iv) Message routing. To reach (i.e., store or obtain) resources in sharing peers, overlay networks must route request messages to peers according to resource indices. If the indices are random, peers may get resource locations from a central server or use broadcast messages in a limited scope to directly query resources. On the other hand, peers can use local information (i.e., routing tables and neighbor lists) to forward the messages toward the peer closest to the resource keys if indices are tight to the structure of the overlay network.

In principle, all peers that form a P2P system should provide their resources to others as well as request services from others, thus forming a *fully decentralized P2P* system. In practice, P2P systems can have exceptions. For example, a *hybrid decentralized P2P* is a system with a central server for some functions: peer discovery, enrollment, or resource indexing, whereas a *partially centralized P2P* system has both client nodes, which generally only request services, and peer nodes (called super nodes), which request/provide services and/or self-organizing functions.

However all advantages (e.g., efficiency, reliability, and scalability) of P2P systems come with security risks since resources in P2P systems rely on multiple trusted/untrusted peers instead of a single trusted server as client-server models do. Without infrastructure, messages in unstructured and structured P2P systems are broadcast/forwarded in the overlay network via intermediate peers, which may be malicious. From these situations, we can classify adversaries into three groups: external, internal, and provider. While external adversaries are any nodes in the underlying networks, internal adversaries are any peers in the overlay networks. Provider adversaries are peers serving required resources. All three adversary groups can launch general network attacks, such as eavesdropping, Man-In-The-Middle (MITM), etc., to disrupt P2P systems. Internal adversaries can launch specific attacks, such as abuse bootstrapping, incorrect routing updates, Sybil, eclipse, identity theft, etc., to subvert overlay network functionalities. Finally, provider adversaries may launch attacks, such as free-riding, whitewashing, etc., related to the resources/contents of P2P applications.

1.2 P2P Security

Although P2P systems gain many benefits from equality in roles, the symmetric roles among P2P peers imposes security burdens because of the lack of hierarchical or central-

ized trust authorities. In the following, we summarize possible attacks, such as abuse bootstrapping, Sybil, eclipse, identity theft, free-riding, and data corruption.

(i) Abuse bootstrapping attacks. Before joining P2P systems, a new node must discover existing nodes called *bootstrap nodes* in the overlay network. The bootstrap nodes will help the new node to find other prospective neighbor nodes. In addition to neighbor lists, this step will allow new structured P2P nodes to create a routing table. Obviously, if bootstrap nodes are malicious or compromised, they can bring the new node to parallel networks instead of real networks or launch other attacks, such as eclipse or identity theft. In this scenario, the adversaries are bootstrap nodes and launch *abuse bootstrapping attacks*.

(ii) Sybil attacks. During registration, P2P systems generate on-line identities, which are random values or hash values of IP addresses, as P2P node IDs for new peers. Thus, adversaries can exploit this vulnerability to generate a number of peer IDs or harvest unused IP addresses for corresponding peer IDs. If the adversaries can obtain enough peer IDs in P2P systems, they can gain advantages from presenting as multiple peers, so call *Sybil attacks*, and thereby launch other attacks, such as eclipse or identity theft.

(iii) Incorrect routing update attacks. In structured P2P systems, when a node joins/leaves overlay networks, it or its neighbors must send a routing update to affected peers in the overlay networks. To convince a target peer to add inappropriate nodes to its routing table, adversaries may send incorrect routing updates to the target peer or cause correct peers around the target peer to fail. Adversaries manipulating a routing table in the wrong direction in *incorrect routing update attacks* can lead to other attacks, such as eclipse or identity theft.

(iv) Eclipse attacks. In overlay networks, peers connect to the networks and communicate with other peers through the links with their neighbor peers. Therefore, if adversaries can control a majority of the linkages of a target node, they can control both the inbound and outbound traffic of that node, in so-called *eclipse attacks*. To launch eclipse attacks in structured P2P systems, adversaries can exploit incorrect routing update attacks to manipulate a large fraction of entries in routing tables in order to control most linkages. Another way to launch eclipse attacks is exploiting Sybil attacks to present as the victim's neighbor peers. After adversaries succeed in launching eclipse attacks, they can censor, replay, or drop the victims' messages as well as disable all services to or from the victims.

(v) Identity theft attack. To reach resources, P2P systems must provide one of three main resource discovery approaches: centralized indexing, limited-scope broadcasting and key-based routing (KBR). Centralized indexing is a simple technique used by hybrid decentralized P2P systems, but it is poor in scalability and reliability. Broadcasting in a limited scope is a common method for unstructured P2P systems, which is more reliable but worse in scalability and efficiency. Key-based routing is the most efficient, reliable and scalable discovery method in P2P systems and is used for structured P2P systems. KBR usually uses a hash function (e.g., SHA-1) to map both peer IDs and resource IDs (keys) into the same large address space (e.g., 160 bits), then places a resource on the live peer (called the key's root node) closest to the resource's key. In the same way, when peers request a resource, the KBR will route to the root node where the resource was stored before. Because the requester does has not have enough information to find the root node by itself, the KBR relies on intermediate nodes recursively querying step-by-step to the root node, then reports to the requester which node is the key's root node. Therefore, for any request to a key, an adversary on the routing path can claim that it is the root node, in so-called *identity theft attacks*.

(vi) Free-riding attacks. Although P2P systems can support routing and resource discovery, the quality of service in P2P systems will be disrupted if most peers use P2P services but do not contribute to routing, storing content, and uploading files, etc., for other peers at an acceptable level. Such selfish behavior, called *free-riding*, can degrade P2P performance significantly, until it becomes non-functional if the fraction of selfish peers is large enough.

(vii) Data corruption attacks. Aside from the quality of service, adversaries may try to corrupt data. If P2P systems lack persistent IDs, authentication and authorization, an adversary can create data with the same key as the existing data and try to store the data in the same root. Without a digital signature to verify data origination, adversaries can still inject faked copies with the same key. Both misbehaviors are called *data corruption attacks*.

1.3 Cloud Computing

Cloud computing is another novel computing system that positions itself between two models: centralized and distributed, in such a way that cloud users see cloud service providers (CSPs) as centralized providers with unlimited pooling resources. Based on distributed and

virtualization technologies, cloud computing can provide scalable and on-demand computing services similar to traditional public utilities, such as electricity, water, natural gas, or telephone networks. With Web technologies and the Internet, cloud computing is a broadly accessible system for client devices and users.

Thus, cloud users can convert capital expenses (e.g., servers, storage, software licenses, etc.) and operational costs (e.g., installation, maintenance, upgrades, retraining, etc.) to on-demand payment (e.g., billed per CPU time, storage space, I/O transactions, units of data transmission, etc.). Without the operation tasks, companies can focus on their business logics as well as quickly and flexibly react to market conditions. For the economies of scale, CSPs pool IT infrastructure together, i.e., hardware, software, and supporting facilities, then provide them to users through the Internet as the following services.

(i) Infrastructure as a Service (IaaS). Generally, cloud hardware is a pool of computing units, storage, and network devices in some forms of distributed system, such as computer clusters, P2P systems, grid computing, distributed storage, key-value storage, load balancing, etc. In multi-tenant models, cloud hardware is virtualized and delivered as services called *Infrastructure as a Service (IaaS)* to IT teams or individual users. Usually, CSPs allow users to manually configure their virtual-hardware capacity via web portals or APIs, while some CSPs automatically provide resizing of capacity based on exhibited load.

(ii) Platform as a Service (PaaS). For cloud software, CSPs can provide services to target users on two levels: development platforms and applications. The development platforms, composed of development tools, computer languages, APIs, and runtime environments, are delivered to application developers as services in a model called *Platform as a Service (PaaS)*.

(iii) Software as a Service (SaaS). CSPs may develop applications by themselves and act as Application Service Providers (ASPs) to offer already-created software for end-users. This service model is called *Software as a Service (SaaS)*.

To sum up, cloud computing can provide three services models, i.e., SaaS, PaaS, and IaaS, which are abbreviated to *SPI model*. The cloud computing systems that are operated by vendors and sold to customers are called *public clouds*. On the other hand, *private clouds* are cloud computing systems solely managed and used by a single organization. Public clouds and private clouds can be bound together in so-called *hybrid clouds* for load balancing or bursting peak load from private clouds to public clouds.

Similar to P2P systems, cloud adversaries can be classified into three groups: external, internal and provider. While external adversaries mean Internet users; internal adversaries mean cloud users who share the same multi-tenant environment. Provider adversaries mean Cloud Service Providers (CSPs) themselves. Both external and internal adversaries can launch general network attacks, such as eavesdropping, Denial of Service (DoS), MITM, etc., in cloud computing through the Internet and internal networks, respectively. In contrast, internal adversaries can exploit vulnerabilities from core technologies or state-of-the-art cloud computing to launch specific or prevalent attacks, such as web interface, cache/memory interference, malicious/illegal images, etc., in cloud computing. Furthermore, CSPs can subvert data security and privacy, i.e., confidentiality, integrity, authentication, authorization, and auditing, because owners lose control of their resources.

1.4 Cloud Computing Security

Despite many promising benefits, users have been quite reluctant to adopt cloud computing for sensitive data and applications due to fear of security threats, privacy risks, and loss of control. Therefore, to make cloud computing use more widespread, we need to understand the vulnerabilities in and possible attacks to clouds. Here, we summarize only the specific attacks that are related to the core technologies or prevalent developments of cloud computing.

(i) Web application and service technologies. All kinds of services in clouds, such as software applications, software development tools, and web portal services, are based on web applications and service technologies. Thus, web vulnerabilities can be exploited by both external and internal adversaries to launch many attacks, such as session riding/hijacking, bogus XML signature wrapping, same origin policy violating, browser authentication replaying/hijacking, etc., so called *web interface attacks*.

(ii) Virtualization. In addition to web technologies, virtualization is another cloud technology that has intrinsic vulnerabilities: *weak isolation* and *image sanity*. For the former, internal adversaries can exploit weak isolation to launch attacks. First, internal adversaries sharing a last-level-cache (LLC) with victims can launch *DoS attack*s to disable the victims' services, or *side channel attacks* to eavesdrop on sensitive information (e.g., secret keys) from a victims' memory. Second, internal adversaries may try to cross-access data belonging to other tenants in the same network storage. For the latter, since cloud images, i.e., Virtual Machine (VM) templates (or implementation modules), are the initial states and security foundations of instances (or

applications), internal adversaries can analyze versions, patches, configuration, histories of OSs (or applications) by renting the same images (or modules) that a target does. Furthermore, if adversaries can bundle malware in cloud images (or modules) and then deploy it in marketplaces, the malware will be mounted on instances (or applications) when victims run such *malicious images* (or develop applications from such *malicious modules*). Third, instead of malware, adversaries may deploy illegal images, i.e., unlicensed or expired-license software, marketplaces, which can bring law-violation changes to either image retrievers or repository administrators.

(iii) IAAA. Generally, Identity management, Authentication, Authorization, and Auditing (IAAA) and identity federation are the main technologies of access control in a single trusted domain. The existing IAAA standards (e.g., OpenID, OAuth, etc.) provide automated user-enrollment, access control, and single sign-on (SSO), as well as outsource processes or data to partners. However, these standards can deploy only in a single trust domain in which the trust boundary, which encompasses systems, networks and applications, is hosted in a private data center and managed by an IT team. Unfortunately, with the adopting of public or hybrid clouds, the trust boundary will become dynamic and beyond the control of an IT team because it extends into a single or multiple CSP domains. This loss of control introduces three crucial open problems, i.e., trust governance, access control, and inter-operation in cloud computing. Thus, cloud computing may not adopt these technologies to provide the same benefits if they are *untrusted/semi-trusted clouds* because the trust boundary splits into two or more domains and neither of them trusts each other. Consequently, the evolution of IAAA technology that can deploy in untrusted/semi-trusted environments is a crucial step for rapid adoption of cloud services. However, these IAAA standards and implementations still have vulnerabilities, such as *weak authentication credentials*, *weak credential-reset mechanisms*, and *coarse authorization controls*, that adversaries may exploit to launch attacks in clouds.

1.5 Motivation

In order to study security in fully decentralized P2P systems and untrusted cloud computing, we first show that they have some common properties in trust and security models.

In terms of trust models, fully decentralized P2P systems have individual trust domains, while both cloud users and cloud service providers have their own trust domains. Thus, neither case has a single shared trust domain.

In terms of security models, adversaries in both systems can be classified into three groups: external, internal, providers.

(i) External adversaries, which are not members of distributed systems (i.e., overlay networks or clouds) can launch only general network attacks from external networks (i.e., underlying networks or the Internet) with limited impact.

(ii) Internal adversaries, which are member of distributed systems, have more impact because they can exploit the intrinsic vulnerabilities of distributed systems to launch specific attacks, including attacks external adversaries can carry out.

(ii) Provider adversaries, such as sharing peers and CSPs, control processes or data in the distributed systems, so they can manipulate processes or data that owners delegate to or store on them. Thus, systems have to cope with the misbehaviors of providers after the owners lose control of their processes or data.

From the summary of trust and security models, it is obvious that we should emphasize the security problems only from internal and provider adversaries and must cope with unshared trust domain problems. Thus, this thesis proposes two contributions that address open security problems in fully decentralized P2P systems and untrusted cloud computing.

For fully decentralized P2P systems, the discussion of P2P applications in Section 2.2 shows that unstructured P2P systems tentatively migrate to structured P2P systems in order to gain more efficiency and scalability. However, structured P2P systems intrinsically have vulnerabilities that impose specific attacks, described in Section 2.6. Previous studies have shown that public-key certificate binding with a node ID and/or an IP address is a necessary security tool for dealing with these attacks. Unfortunately, a traditional Trusted Third Party (TTP), such as a Certificate Authority (CA) or a Public Key Infrastructure (PKI), for certificate issue is not suitable for fully decentralized P2P systems, which are self-organizing and infrastructureless. Therefore, solving unshared trust domain problems and automating public-key certificates issue in P2P systems for peer enrolment (registration) are challenging security issues in P2P systems.

In untrusted cloud computing, the cloud architecture, as described in Section 2.7.4) points out that IAAA is the core-technology leveraged by CSPs for access control, operational security (e.g., enforcement of compliance requirements and assignment of limited privileges), and inter-operation. However, the vulnerabilities studied in Section 2.10 show that current IAAA technology has vulnerabilities in both authentication and authorization steps that adversaries can exploit to launch specific attacks, as presented in Section 2.11.

Moreover, three existing standards, i.e., Kerberos, OpenID, and OAuth, described in Subsections 3.2.2, 3.2.3, and 3.2.4, are based on a single trusted domain. Therefore, an authorization scheme for multi-trusted domains and semi-trusted clouds is still an open problem for inter-operation in cloud computing.

1.6 Thesis overview

This thesis presents two contributions for the security in distributed systems.

1.6.1 Self-Organizing and Self-Healing CA Group (SOHCG)

As noted in the survey of P2P systems in Section 4.1, public-key certificates for P2P systems are managed in three main approaches: centralized/hierarchical (CA or PKI), distributed (PGP), and individual. For a PGP model, Takeda *et al.* [125] proposed a Hash-based Distributed Authentication Method (HDAM), which uses a Distributed Hash Table (DHT) to maintain a certificate database in P2P systems, and uses a PGP model to create trust relations from mutual authentication between two nodes along the lookup path in a P2P network. Usually a certificate can be looked up and endorsed in $O(\log_2 N)$ steps. The main drawback of this system is that only one malicious peer on a route path can subvert a trusted relation in PGP models. For an individual model, V. Pathak and L. Iftode [96] proposed Byzantine fault tolerant public-key authentication, which can prove the possession of a private key under an honest majority. This approach does not require TTPs, and the authentication is correct if no more than $\lfloor \frac{n-1}{3} \rfloor$ of the n parties in individual trust groups are malicious or faulty. The main drawback is that the trust group and the proof of private-key possession are under only an individual peer's scope. Hence, trust relations cannot be transferred to other peers in the P2P system, thereby causing burden load for every peer in the system. For PKI models, Avramidis *et al.* [9] proposed Chord-PKI, which is a distributed PKI embedded into the Chord overlay network in order to provide certificates without external PKI. The main idea is to partition the Chord network into multiple areas and empower some trusted nodes in each area for certification functionalities, and to employ a (t, n) threshold signature to sign the certificate by the coalition of the trusted nodes from each area. However, since the members and number of a trusted group are static, the trusted nodes are targets for adversaries wishing to subvert this distributed PKI. Lesueur *et al.* proposed another distributed PKI system based on the trusting of $t\%$ of nodes in a P2P system. Thus, a certificate must be signed by the collaboration of $t\%$ of nodes. To maintain the ratio of $t\%$ of nodes in a collaboration, the

number of secret shares must increase according to the increase of the number of nodes in a P2P network.

Although the previous works try to provide public-key certificates in P2P systems, none of them can achieve our three design goals:

(i) Self-organizing approach. A framework should be able to delegate a trusted group to manage public-key certificates without an external CA. After a bootstrapping phase, the trusted group should be able to maintain its size and membership by itself. In the running phase, the trusted group should automatically identify a node ID and verify private-key possession before issuing certificates to new nodes.

(ii) Self-Healing approach. The trust group itself must be able to detect misbehavior of its members to maintain an honest majority in a trusted group. Each member should have a limited lifetime to join the trust group. The trust group must automatically refresh or update secret shares.

(iii) Scalability approach. The framework must not cause significant load for large scale networks like the Internet.

Therefore, we propose the Self-Organizing and Self-Healing CA Group (SOHCG), which can achieve our design goals. A detailed design of the framework is describe in Chapter 4.

1.6.2 OAuth and ABE Based Authorization (AAuth)

The survey of previous works in Section 5.1 shows that researchers first focused on secure and authentic distributed file systems in order to outsource storage for some specific data, such as archive files, backup files, or scientific data, etc. Next, the researchers proposed key management systems for public clouds in such a way that the owners encrypt and sign their data before the data is stored in cloud storage, and encrypt the keys used to decrypt data with users' public keys. To retrieve data, users first use their private keys to authenticate themselves to and decrypt the keys from the key management system. Then users use such keys from the key management system to decrypt the data stored in cloud storage. In other words, previous works use public-key certificates as credentials for access control of key management systems. To our knowledge, there is no generic scheme or standard that supports data owners and Application Service Providers (ASPs) in performing inter-operations with untrusted cloud storage without losing control of their data.

Our solution is to extend the OAuth standard by using ABE-tokens to establish an abstract layer that separates authentication from authorization without restrictions to any user credentials and delegates both authentication and authorization from owners to consumers with end-to-end approaches. Our scheme puts no key management systems in CSPs, rather it constructs ABE-tokens from the cooperation of existing cloud entities and owners with user-centric approaches and the distribution of key knowledge to minimize risks in semi-trusted cloud environments. Thus, our scheme is more abstract and generic than others. The details of our AAuth authorization scheme are described in Chapter 5.

The remainder of this thesis is organized as follows. Chapter 2 is divided into two parts: P2P systems and cloud computing. The former gives the basic concept, applications, and threats to P2P systems. The latter gives the basic concept, definition, vulnerabilities, and threats to cloud computing. Chapter 3 gives the background and definition of security technologies used in this thesis. In Chapter 4, we describe our security framework (SOHCG) for P2P systems. First, we summarize previous work related to security in P2P systems. Second, we present system and security models of SOHCG. Third, we describe SOHCG construction and its protocols. Last but not least, we analyze the security of the SOHCG framework. In Chapter 5, we describe our authorization scheme (AAuth) in cloud computing. First, we present previous work related to cloud storage and access control. Second, we introduce the AAuth system and adversary models. Third, we give the definition and notation of AAuth construction. Fourth, we describe AAuth procedures and protocols. Fifth, we analyze the security of the AAuth scheme. Last but not least, we evaluate the AAuth scheme by simulation. Finally, we conclude the thesis and identify areas for future work in Chapter 6.

Chapter 2

Preliminary

2.1 Basic Concept and Applications of Peer-to-Peer (P2P) Systems

The Peer-to-Peer (P2P) system was initially developed from a content-distribution application, music file sharing, that had no security provisions. In the middle of 1999, Napster [86] was introduced, and there were about 50 million users by December 2000. Although Napster uses peer-to-peer communication for actual file transfer, it needs a central server to store the index of files in the Napster community. Consequently, its centralized index is a single point of failure and a limitation for scalability. The following generation of P2P systems, e.g., Freenet [26], Gnutella [107], and Kazaa [72], are unstructured and decentralized mesh networks. These P2P systems flood request messages within a certain scope. As a result, they are not scalable and may fail to find a required file. The third generation, a structured P2P system is much more scalable, reliable and efficient than its ancestors because it uses a structured manner to route messages and place objects. Therefore, a structured P2P system can support much more complex applications, such as storage management, service discovery, application-level multi-cast, data objects location (web caches), publish/subscribe and decentralized spam-filtering. Based on their rapid developments and dramatic increase in the number of users, we believe that the new applications and communications leveraging of these P2P systems will necessitate effective security measures, i.e., access control, authentication, data integrity and non-repudiation.

2.2 P2P Applications

With the revolution of client-server applications, P2P applications have evolved from file sharing and instant messaging applications into many Internet applications. The key reasons for using P2P architecture instead of client-server architecture are to achieve reliability, efficiency, and scalability. Commonly, P2P applications enable users to share computing power, data and network bandwidth (i.e., using many nodes for transferring data). Therefore, many P2P applications, e.g., collaborative work, Voice over IP (VoIP), video streaming, and distributed file systems, etc., have emerged. Although many companies, such as Sun, Microsoft, Intel, IBM, etc., are researching their own standards, no single acceptable standard for P2P applications has been established. In the following subsections, we present some common P2P applications by giving popular examples and relevant research work.

2.2.1 P2P File Sharing

The most popular P2P file sharing applications [64] are Guntella, FastTrack, eDonkey and BitTorrent.

Guntella: was developed by Justin Frankel and Tom Pepper from Nullsoft in 2000 [107] [82]. Gnutella is a purely decentralized and unstructured P2P file sharing application that became well known during Napster's legal troubles in 2001. A new node joins the network via a bootstrap node whose IP address is published in the pre-configured list of client software or a public server, i.e., a web server, a UDP host, an IRC server, etc. When a new node participates in the network, the new node discovers more existing nodes until the number of neighbors reaches a predefined threshold. To query a target file, a query message is propagated to the Gnutella network within a limited number of hops. A later version (Guntella v2) was implemented in a partially centralized architecture that consists of super nodes and leaving nodes.

FastTrack: was introduced by Niklas Zennstrom, Janus Friis and Jaan Tallinn in 2001 [75]. There are two worthwhile features in FastTrack. First, it uses super nodes to improve scalability. Second, FastTrack employs UUHash, the 160-bit hash function that concatenates 128-bit and 32-bit hash values from a file together. The 128-bit part is generated by using a MD5 hash function for the first 300 kilobytes of a file, and the other 32-bit part is generated by applying a small hash function to 300 KB blocks at file offsets 2^n MB with integer n incremented from 0 until the offset reaches the end of the file, This technique can find

identical files in multiple locations, so it enable one to download a file from multiple nodes simultaneously.

eDonkey: was developed by Meta Machine Inc. in 2000 [52]. eDonkey is a hybrid decentralized P2P system like Napster. Therefore, the eDonkey server functions has a communication hub with index files and an address directory for clients. A later version, eDonkey v2, was implemented in a partially centralized architecture that uses the Kademlia overlay network (see Appendix A.4) to overcome the problems of overloading and security attacks.

BitTorrent: was developed by Bram Cohen in 2002 [100]. To share a file through a BitTorrent network, the Torrent file, which consists of tracker information and the hash of a file block, is published on a web server, a forum, a BBS board, etc. When a client wants to retrieve a file, it first obtains the Torrent file, then extracts the tracker(s) to obtain a list of nodes sharing the file. BitTorrent has three beneficial features that make it popular for file sharing. First, it employs Tit-for-Tat and optimized unchoking strategies to defend against free-Riding attacks. Second, the hash of a file block in a Torrent file can protect against pollution attacks. Third, splitting a file into blocks and sharing a file block on the basis of rarest locality can utilize bandwidth effectively.

2.2.2 P2P Communication and Collaboration

This category of applications includes systems that provide infrastructures for facilitating direct and real time applications and collaboration between computer peers. We can classify this category into three main groups as follows.

Instant Messaging (IM): emerging in the early stages of the Internet, is based on pure client-server architecture, such as IRC [66] [67]. Currently, Instant message applications, such as AOL [5], ICQ [58], Yahoo [131], MSN [85] and Jabber [61], are implemented by a hybrid decentralized P2P system that uses a server or distributed servers to be a broker of user information and presence. However, to support millions of users in the Internet scale, researchers seek to integrate user presence and communication within their own application. Therefore, the system operates as a purely decentralized P2P system, thereby eliminating all infrastructures. To accomplish this goal, in 2007, Ravi and Sandeep proposed P2P-IM [105] based on a lightweight presence system. The foundation of the P2P-IM presence system is a user's *personal identity*, which is a locally-generated public/private key pair associated with a self-signed certificate. In addition, each P2P-IM user maintains a *trusted*

contact list, which is a list of personal identities of other trusted users. To immediately obtain presence information when other users come online, P2P-IM lookup must operate in real time and be authenticated to avoid attacks. This lookup and authentication use P2P SDK, which is based on a name resolution protocol and a special Distributed Hash Table (DHT). Moreover, P2P-IM supports standard features like a traditional IM application, such as Multi Point of Presence (MPOP) and presence status (On-line, Busy, Away, etc.).

Collaborative work: is an application that helps a team work together dynamically and efficiently. This collaboration can support a team that works remotely, off-line or from different organizations. In the early stage of collaborative software, it was implemented by an Internet portal and a collaborative server. However, updating information with portals is not always possible, for instance, in off-line and ad hoc operations. Therefore, the next generation of this application uses a P2P system that can break this barrier. A P2P collaborative software should provide user and role based security, persistence storage, fault tolerance and user presence awareness, thereby supporting across a large number of users. For example, Groove acquired by Microsoft in April 2005 is software with P2P capabilities for document collaboration and instant messaging, and includes video conferencing.

Skype: was developed by Nikle Zennstrom and Janus Friis, then acquired by Microsoft in May 2010. Instead of using standard protocols (SIP and H.263), Skype [64] uses a proprietary protocol (based on Kazaa) and a partially centralized P2P system that consists of super nodes and clients. Skype has three main features that differ from those of other typical VoIPs and P2Ps. First, it can detect the existence of a firewall/NAT to ensure that the Skype connection can be established either directly or indirectly. Second, it automatically checks the resources (e.g., CPU, memory, network bandwidth) of Skype nodes. Third, it uses the public-key cryptographic functions for call authentication and Advanced Encryption Standard (AES)for call confidentiality. The powerful and directly connected (i.e., not behind NATs or firewalls) nodes are promoted to be super nodes that perform user directory services and relays traffic for computers behind the firewall/NAT.

2.2.3 P2P-SIP

IETF's Session Initiation Protocol (SIP) is a standard protocol for Internet telephony client-server architecture. Since this model needs maintenance, configuration and a dedicated server, people have proposed a P2P architecture for SIP-based telephony systems to overcome these drawbacks. There are two approaches to combine SIP and P2P. First, the SIP-P2P approach is implemented by replacing the SIP user registration and lookup with

an existing P2P lookup. Second, the P2P-SIP approach implements the P2P lookup protocol with standard SIP messaging. In 2005, Sing and Schulzrinne [116] proposed a P2P-SIP system that is based on the second approach and Chord (see Appendix A.1) as the underlying DHT. The main characteristic for their P2P-SIP is a purely decentralized model that increases robustness, cooperation with existing infrastructures (e.g., DNS, SIP-voice mail service, SIP-PSTN gateway, etc.), and independence from external P2P networks. The P2P-SIP now depends on a DHT structure. Hence, this P2P-SIP needs security provisions that can protect the DHT structure.

2.2.4 P2P Streaming

The basic solution for video streaming over the Internet is a client-server architecture and its variant, Content Delivery Network (CDN). In a CDN solution, a video server pushes video stream to a set of CDN servers that are placed across the whole network strategically. Clients directly download from a nearby CDN server instead of the video server. Thus, the CDN servers can shorten delays and reduce the amount of traffic in the network. For example, *Youtube* employs CDN servers to stream video to end users. However, the bandwidth at a video server and CDN servers must grow proportionally with the client population. The alternative solution is an IP-level multi-cast, which is an effective way to stream audio or video in the Internet. Unfortunately, IP-level multi-cast is rarely deployed on the Internet because its routing overhead and complexity of traffic control are too high. In contrast, a simple multi-cast function can be implemented at the application-level efficiently and economically. Therefore, video streaming over a multi-cast overlay network [76] is a new paradigm for this application.

In P2P streaming, every node is encouraged to act as both a client and a server, namely a peer. Therefore, a peer not only downloads video stream from a network but also uploads the video stream to other peers in the network. This uploading bandwidth from peers replaces the required bandwidth at video and CDN servers. The P2P streaming system can be classified into two categories based on the overlay network structure: tree-based and mesh-based.

The tree-based structure progressively pushes video stream from parent peers (note that a root peer is a video server) to child peers. A simple solution is to form a single tree whose root is a video server at the application-level, namely a single-tree structure. For instance, Overcast and ESM P2P streaming are based on a single tree structure. The disadvantage of a single tree structure is that all leaf nodes do not contribute their uploading bandwidth. Since the number of leaf nodes constitutes a large portion of the peers in the system, the

bandwidth utilization is downgraded significantly. To address this drawback, a multi-tree structure is proposed to upgrade the bandwidth utilization. In a multi-tree structure, a video server divides the stream into m sub-streams, which then are pushed into m sub-trees, one for each sub-stream. Each peer may be placed on an internal node of one sub-tree and on a leaf node of other sub-trees. By this strategy, the uploading bandwidth of a peer is proportional to the number of sub-trees in which the peer is an internal node.

However, the tree-based structures have only one parent in a single-tree structure or a sub-tree of a multi-tree structure. This constraint is a single point of failure if the system faces a high churning rate. Therefore, many P2P streaming systems adopt a mesh-based structure, with a dynamic topology. Each peer in a mesh-based structure maintains connections with multiple peers based on the content and bandwidth availabilities on the other peers. Hence, video content is simultaneously downloaded from multiple peers who have already obtained the content. A mesh-based structure is robust against peer churns. However, the dynamic topology causes the video distribution rate to be unpredictable. Consequently, the system may suffer from long delays, freezing and low quality playback.

2.2.5 P2P Distributed File Systems

Although a file system provided by a central storage system is efficient for basic data operations (insert, delete and query), experience shows that a distributed approach is better for achieving reliability. In initial efforts, a client-server architecture is employed for caching, replication and availability. As the Internet grows, a distributed file system needs new criteria, such as availability, fault tolerance, security, robustness and locality. Inherently, P2P architecture can reduce storage and bandwidth costs and allow cost sharing by existing infrastructures from different nodes. Based on the aspects of P2P systems, a P2P file system [53] can provide many beneficial properties, such as load balancing, scalability, anonymity and persistence of information. However, a practical P2P file system that conforms to all of the previous properties rarely exists. We present some of these systems as follows.

FreeNet: [26] is an adaptive P2P file system that operates as a location-independent distributed file system across many individual computers and allows files to be operated anonymously. FreeNet uses probabilistic routing to publish, replicate, and retrieve information, meanwhile preserving the anonymity of the author, reader and data location. The design goals of FreeNet are anonymity, deny-ability, dynamic storage, dynamic routing, resistance to third-party access, and decentralized policies.

Cooperative File System (CFS): [30] is a P2P file system developed at MIT with design goals of provably guaranteeing efficiency, robustness, load balancing, and scalability. CFS divides a file into constituent blocks that are stored in different nodes over the Chord (see Appendix A.1) overlay network, which maintains lookup and query management. However, CFS is a read-only system for a user. A publisher can update information by using a key-based authentication.

PAST: [109] is a large scale P2P persistent storage management system based on the Pastry (see Appendix A.2) overlay network. PAST is composed of storage nodes, which query files in a cooperative manner and perform replica storage and caching. In PAST, a file ID is a SHA-1 hash value of the file name and the public key of the client. Hence, Files in PAST are immutable, that is, multiple files cannot have the same file ID.

OceanStore: [70] provides a persistent storage for nomadic data in a uniform global scale that is based on the Tapestry (see Appendix A.5) overlay network. OceanStore is designed for a scenario in which providers form agreement and resource sharing, and consumers pay fees to access the persistent storage. To accomplish these goals, OceanStore caches data in the network with encryption, uses ACLs for restricting write access to data while using keys for read access and, uses a Byzantine agreement algorithms between primary and replica nodes for updating data.

Farsite: [2] is a symbiotic, symmetric, and distributed file system; which works cooperatively but does not trust other nodes completely. It was designed to provide highly available, reliable and secure file storage, and resistance to Byzantine threats. Farsite encrypts contents to prevent unauthorized reading and digitally signs contents to prevent unauthorized writing. Moreover, it has a collection of Byzantine-fault-tolerant replica groups arranged in a tree-based structure of the file name space to resist arbitrary faults.

2.2.6 Summary

From the preceding of survey on P2P applications, we can conclude that many Internet applications are moving from a client-server model to a P2P model. Meanwhile, both hybrid decentralized and unstructured P2P systems are migrating to either partially centralized or purely decentralized structured P2P systems.

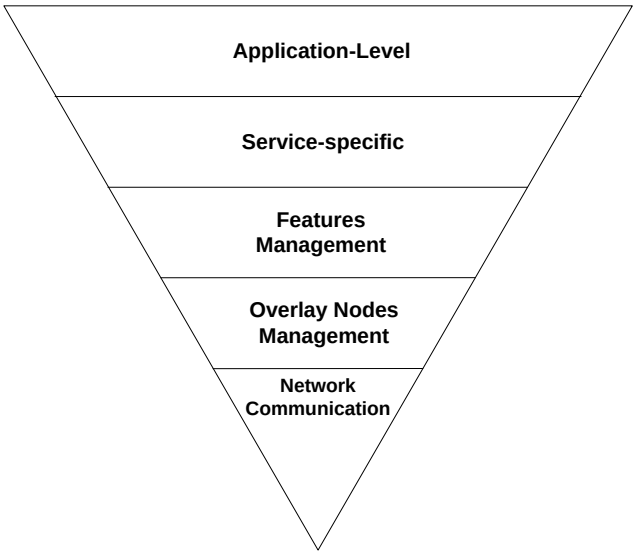

Figure 2.1: The abstract layers of P2P overlay network architecture

2.3 Structured Peer-to-Peer Systems

A P2P network uses a distributed algorithm that takes responsibility for locating content and nodes. This algorithm is implemented in an application layer separated from the routing mechanisms in a network layer. Therefore, a P2P network is an overlay network that runs and spreads on existing communication systems, e.g., the Internet, wireless networks, etc. Intuitively, a P2P network is a decentralized and self-organized system that provides many positive attributes: availability, reliability, load-balance, fault-tolerance, scalability and efficient data management. The abstract layers of P2P overlay architecture taken from [78] is shown in Figure 2.1, which are described below.

Application-level: consists of tools, applications and services that are implemented on top of P2P infrastructures.

Service-specific: provides task/service scheduling, content/service management, messaging, and meta-data.

Features management: facilitates security, reliability, fault resiliency, and resource management.

Overlay nodes management: conducts routing, lookup, locating, and resource discovery.

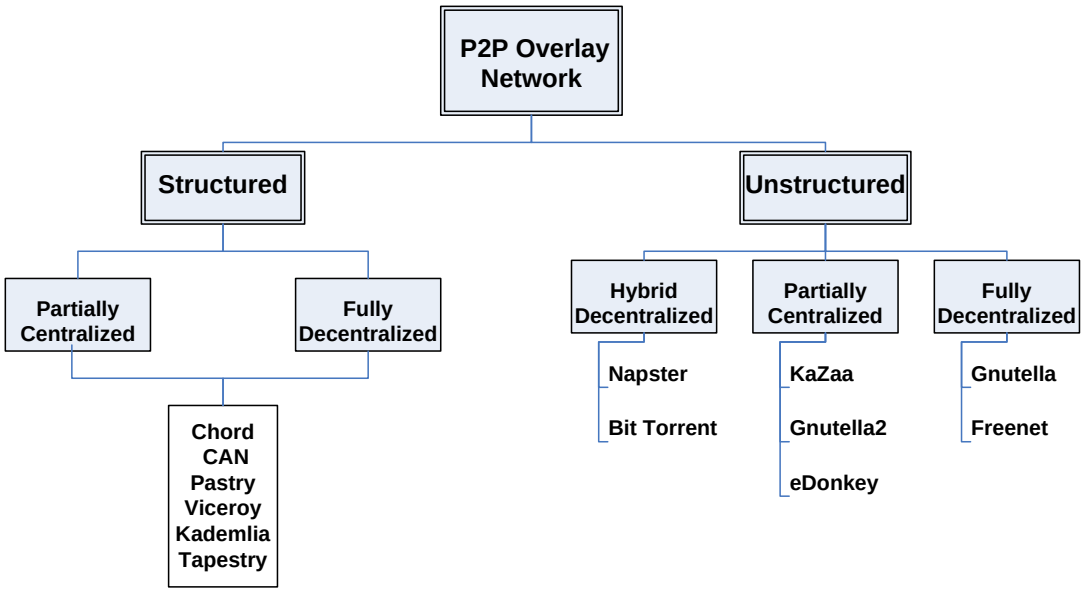

Figure 2.2: The taxonomy of P2P systems

Network communication: consists of Internet, wireless, an-hoc, or sensor networks.

P2P systems can be classified into two types: structured and unstructured overlay networks. In a structured P2P system, content is placed on the topology of the overlay network tightly, whereas the content is placed randomly in an unstructured P2P system. Both structured and unstructured P2P systems use a Global Unique Identifier (GUID) in a single identifier space to identify a node object and a content object. We define the identifier of a node object as a **node ID** and the identifier of a content object as a **key**. Moreover, we can classify P2P systems according to ascending degree of decentralization: hybrid decentralized, partially centralized and purely decentralized P2P systems. A hybrid decentralized P2P system has a central server maintaining an index of location information. A partially centralized P2P system uses a group of super nodes to provide services or to maintain an index. In a purely decentralized P2P system, every node, called a **servent**, both requests and provides services with other nodes. The taxonomy of P2P systems according to the degree of structure and distribution are shown in Figure 2.2.

In general, a structured P2P system employs a Distributed Hash Table (DHT) to map a content object (key) to a particular location (node) of an overlay network. Based on localized information and a progressive manner, each node maintains a routing table whose entry consists of a neighbor's node ID and its IP address, and uses this routing table

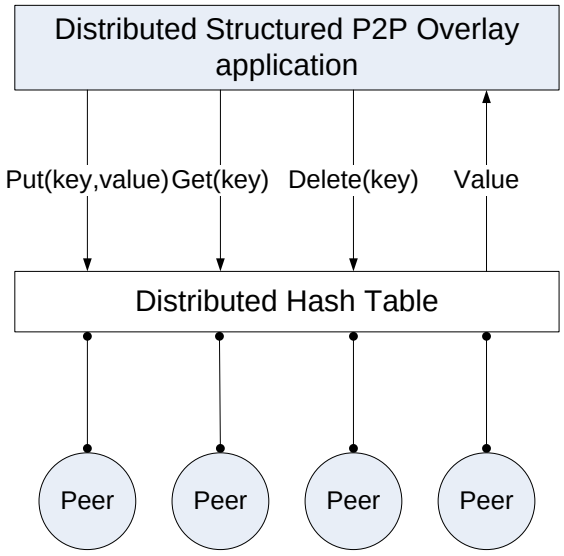

Figure 2.3: The API for a structured P2P system

to route a message to the next closest node node-by-node until the message reaches the destination node. As a result, a structured P2P system can guarantee that any content objects will be operated, i.e., put, get and delete, in the upper bound of a route path length. All of these functions are provided by Application Programming Interfaces (APIs) for application-level independence. The APIs are developed from the emergence of the middle-ware layer on the global scale of P2P systems, as shown in Figure 2.3. For the example of file sharing systems, a file name is input to a hash function which outputs a key, then the file (value) is stored at the key location, i.e., the node whose ID is closest to the key. This step is achieved by the API $put(key, value)$ that routes to the appropriate node for storing this file, then stores the key and value in that node. When other nodes need to obtain this file, they hash the file name to be the key and then use the API $get(key)$ to obtain the value associated with the key from an appropriate node. Similarly, this file can be removed from the designated node by the API $delete(key)$. Although structured P2P systems efficiently locate any content objects, they incur significant overheads to maintain their routing tables and structures. Furthermore, the lower number of hops on overlay networks cannot guarantee lower lookup latency because the underlying networks can be significantly different from the overlay networks.

Without topology relations, unstructured P2P nodes join and leave the overlay network subject to only loose rules. To insert, retrieve and remove content objects, a node uses a flooding mechanism with a limited scope to send query messages across the overlay

network. Although a flooding technique is effective for locating highly replicated items, and resilient to node joining and leaving, the performance of flooding techniques is poor for locating rare items. Moreover, an unstructured P2P system is not scalable and cannot guarantee the upper bound of route path length to access a content object. Even though both structured and unstructured P2P systems are robust, they still suffer from many possible attacks.

In the following sections, we discuss the Content Addressable Network (CAN) structure, which is related to this thesis. Furthermore, we review other significant structured P2P systems: Chord, Tapestry, Pastry, Viceroy and Kademlia.

2.4 Content Addressable Network

A Content Addressable Network (CAN) [103] is a distributed, Internet-scale, hash table overlay network. CAN's identifier space comprises the points in a d-dimensional Cartesian coordinate space on a d-torus. This space is dynamically partitioned into n areas called zones, where n is the number of all current nodes in a CAN. Each zone is split from an existing zone and assigned to a new node when it joins a CAN. When an existing node leaves, its zone is handed over to its neighbor node. Therefore, there is no unassigned zone in the system, and each node maintains the identifier space covered by its zone. The coordinates of a zone are used to route to the node occupying the zone associated with the key by a greedy routing strategy. In other words, each node maintains a coordinate routing table of zones adjoining its own zone and uses this table to route to arbitrary points in the CAN by forwarding to the immediate zone closest to the destination point until it reaches the zone on which the point lies. As a result, the d-dimensional CAN must maintain $2d$ entries of the coordinate routing to guarantee that the average path length is $O(d \cdot n^{1/d})$, where n is the number of nodes participating in an overlay network. For example, for a 2-dimensional CAN, Figure 2.4 shows node 1's neighbor nodes, $\{2, 3, 4, 5, 6\}$ and the routing path from node 1 to a point (x, y). After node 7 joins, Figure 2.5 shows that the zone of node 1 is split: in this case, node 1's neighbor nodes are $\{2, 3, 4, 7\}$, and node 7's neighbor nodes are $\{1, 2, 5, 6\}$.

To look up the content, the content identifier is converted to a key (point) via a hash function and used as the destination point for routing to the node keeping the $(key, value)$ pair. The new $(key, value)$ pair of the content can be stored in the node occupying the zone on which the point (key) lies by API $put(key, value)$. Moreover, an existing value can be obtained by API $get(key)$. To join a CAN, a new node p looks up the CAN domain in DNS, which returns a bootstrap node's IP address. The bootstrap nodes, which maintain

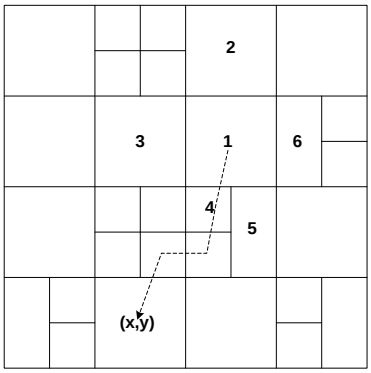

Figure 2.4: A routing path on a 2-dimensional CAN

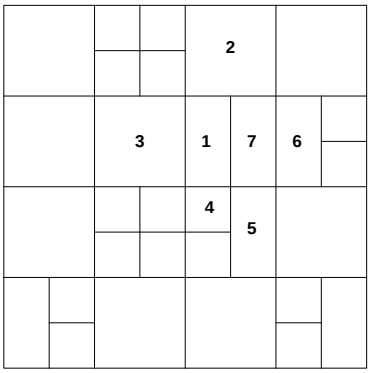

Figure 2.5: A zone spiting on a 2-dimensional CAN

a partial list of CAN nodes, randomly choose a node a from the list and provide the node p the IP address of node a. Then, node p selects a random point α, and sends a join message though node a. The join message is routed to the node b occupying the zone on which the point α lies. Next, node b splits its zone, maintains a half and hands over the other half to node p. In order that the split zone can be merged as the same, splitting is done by a certain ordering of dimension, e.g., for a 2-dimension CAN, starting from X-dimension, then Y-dimension and so on. In addition to splitting a zone, a half of $(key, value)$ pairs held by node b are handed over to node p as well. Node p must learn the IP address of its neighbors from node b's routing table, and node b itself must update its routing table also.

When a node q leaves a CAN, node q will select its immediate zone occupied by node c, then hand over its zone and $(key, value)$ pairs to node c. Then, node c will merge its zone with node q's zone. Generally, a CAN's average path length increases as the number of nodes ascends and as the number of dimensions descends. Moreover, a CAN has two main strong points: a) durability under node churning, i.e., only $2d$ nodes are affected by joining or leaving a node, b) scalability, i.e., the size of a routing table is independent of the current number of nodes in the CAN system and fixed at $2d$ entries. Many techniques have been proposed to improve CAN's properties: gaining performance and reliability, reducing path length and lookup latency, and increasing routing tolerance and data replication. However, these improvements are traded off with the increasing sizes of the routing table and peer list in each node. Some variants of CAN, i.e., multi realities, overloading zones and multiple hash functions, can provide data replication in CAN.

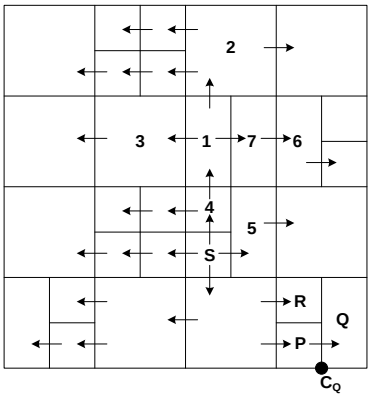

Figure 2.6: The flooding over 2-dimensional CAN

2.5 CAN-Based Multi-Cast

CAN-based multi-cast [104] is an application-level multi-cast group over the CAN overlay network. If all nodes of a multi-cast group form a CAN overlay network, multi-casting a message is achieved by flooding the message over the reforming CAN network. In other words, multi-casting a message over the CAN network is performed by two steps: forming a "mini" CAN over the underlying CAN, followed by flooding the message over the mini-CAN. Assuming that there is a subset of nodes in an underlying CAN that wish to join a multi-cast group, the members of the multi-cast group g then form a mini-CAN C_g over an underlying CAN C. To form a mini-CAN C_g, the multi-cast name of 'G' is hashed by a known hash function to a point β, which is the multi-cast address of G. Hence, the node b holding the point β serves as a bootstrap node of the mini-CAN C_g. Now, joining a multi-cast group G is reduced to joining a mini-CAN C_g, which is done by the original CAN construction process. Flooding over the CAN structure can be efficiently done by exploiting the coordinate space structure of d-dimensional CAN, as shown in Figure 2.6 and described in the following steps:

1. The source node forwards a message to all its neighbors.
2. A node receiving a message from its neighbor in dimension i forwards the message in both forward and reverse directions for dimensions $1, \ldots, (i-1)$, and in only a forward direction for dimensions $i, \ldots, d$.
3. In each particular dimension, a node does not forward a message after the message has already traversed at least half-way along the dimension from the source coordinate.

4. Each node caches the sequence number of messages it has received and does not forward a message that it has already received.

This flooding algorithm ensures that no node has a duplicated message under a perfectly partitioned coordinate in which every node occupies an equal-sized zone. However, for imperfectly partitioned space, it is easy to avoid a duplicated message because every CAN node knows the coordinates of its neighbors. For example, in Figure 2.6, if node Q abuts node P in the forward direction of dimension 1 and the corner C_Q of node Q that abuts node P along dimension 1 is the lowest coordinates of all other dimensions, then only node P forwards the message to node Q, but node R does not forward. That is, we can apply certain deterministic rules with the knowledge of the coordinates space to eliminate duplicates that arise with flooding along the first dimension. Although only all duplicates in the first dimension can be eliminated, most flooding occurs in the first dimension, so most duplicates are eliminated from multi-casting. In this way, a CAN can support an application-level multi-cast service efficiently without the need to compute spanning trees for every source of a multi-cast group.

2.6 Threats to Structured Peer-to-Peer Systems

Commonly, a structured P2P system is built by a DHT structure, which is maintained by the cooperation of nodes in such a system. The benefits from this DHT structure are that nodes in a structured P2P system can look up keys, provide persistent storage, and support multi-cast services efficiently. Therefore, many network applications based on a structured P2P system can obtain these benefits if nodes work together correctly and honestly. Unfortunately, most P2P systems are developed over untrusted underlying networks such as the Internet, in which nodes may be malicious. These malicious nodes may attack P2P systems to gain their advantages, thereby incurring many attacks in P2P systems. Some attacks are general network attacks, such as Man-In-the-Middle(MITM), DoS, Sybil, etc. The other attacks are only launched in P2P systems, such as free-riding, whitewashing, abuse bootstrapping, incorrect routing updates, eclipse, ID theft, etc. The following subsections focus on the main attacks that try to disrupt or subvert structured P2P systems.

2.6.1 Abuse Bootstrapping Attacks

To participate in a structured P2P system, a new node must first contact an existing node, named a **bootstrap node**. If a bootstrap node is malicious or compromised, a new node will connect to the malicious node in a joining procedure, thereby causing traffic censoring and denial of service. Sit and Morris [118] described how a new node is vulnerable to partitioning into a parallel network. Suppose that a set of malicious nodes colludes to form a parallel network running the same protocol as the legitimate network runs. One of the malicious nodes is a bootstrap node and cross-registered between a legitimate network and a parallel network. If a new node accidentally joins such a bootstrap node, the new node may be diverted to the parallel network instead of the real network. Hence, the malicious node can deny the services or learn the behavior of a victim. Sit and Morris also proposed a solution in which each node maintains a set of other nodes that it previously succeeded in connecting with, and then randomly queries via the nodes in this set. The comparison between the present and previous query can verify whether the view of the network is consistent. However, this solution is not suitable for high churning P2P networks and networks in which nodes' IP address are assigned via DHCP. The final suggestion is to use a **certificate with a node ID and a public key** to safely join the system.

2.6.2 Incorrect Routing Update Attacks

In a structured P2P system, nodes must update their routing tables and neighbor lists because of churning. When a node joins/leaves the network or discovers other nodes joining/leaving the network, that node must notify its neighbors to update their routing tables and neighbor lists. Then the notified nodes must add a new node to their routing tables or replace the leaving node with a neighbor of such a leaving node. However, an attacker can abuse this process in two ways. Firstly, an attacker can send out an incorrect routing update to attract other peers to add an inappropriate node or a non-existing node. Secondly, an attacker can cause a target node to fail, thereby convincing other nodes to select an inappropriate node to be an intermediate node.

To cope with this attack, Sit and Morris [118] suggested that routing entries in P2P systems should have certain requirements that are able to verify the correctness of update information. For instance, Pastry (see Appendix A.2) updates require that each table entry has a correct prefix, each CAN routing table entry is a neighbor in each dimension, and each routing table entry of Chord (see Appendix A.1) is close to a specific point. Furthermore, the IETF network working group [79] suggests that the notified nodes should detect incorrect routing updates by contacting the reported leaving node. If the routing

updates originate from the leaving node itself, the notified nodes must verify the node ID of the notifier to ensure that the message is not forged. Additionally, update messages should have integrity protection to prevent being replayed and modified. Thus, the solution for addressing this attack is to use **constrained routing tables** and a **digital signature** to sign the routing updates.

2.6.3 Sybil Attacks

Typically, identity assignment in P2P systems has two approaches: firstly, a new node randomly choses its own identity in a P2P address space, such as, Pastry (see Appendix A.2) and CAN (see Section 2.4), etc. Secondly, an identity is assigned by a cryptographic hash of a node's IP address, such as in Chord, Tapestry, etc. (see Appendix A.1 A.5). Therefore, a small number of malicious nodes may counterfeit a large number of identities by arbitrarily choosing multiple node IDs or spoofing IP addresses that hash to target node IDs. In this way, an attacker can obtain multiple identities and join a P2P system as multiple nodes thereby launching a Sybil attack in the P2P system. In 2002, Douceur [38] argued that without a logically trusted identification authority to vouch for one-to-one correspondence between an entity and an identity, it is practically impossible to prevent an entity from presenting as more than one identity.

To address this attack, Castro *et al.* [21] proposed two solutions. In the first solution, an identity registration with Certificate Authority (CA) must be used to generate a public-key certificates binding to a node ID and/or IP address. For doing so, the registration must be controlled tightly by checking for forged node IDs or spoofed IP addresses. Unfortunately, the CAs present a single point of failure and contradict the idea of purely decentralized P2P systems. Another solution is that an identity-generation process forces nodes requesting new identities to compute difficult tasks, such as solving cryptographic puzzles, thereby no requirement for a centralized CA. However, the cost of solving puzzles must be acceptable for legitimate nodes and hard enough to slow down attackers. In addition, if an attacker has enough resources and time, it can still obtain multiple identities. Finally, a long-term identity, such as a **certificate** binding a node ID to a public-key, may be considered as a security foundation to provides message authentication and integrity and to allow a new nodes to joint the system safely.

2.6.4 Eclipse Attacks

Each node in an overlay network must maintain an overlay link to a set of intermediate nodes for forming an overlay network and routing to the destination. The operations over overlay networks rely on the assumption that correct nodes must forward messages to appropriate nodes along overlay links. If attackers can control a large fraction of links to a correct node, they can eclipse (i.e., censor, replay, or drop messages, etc.) the correct node and prevent correct overlay operations (i.e., deny services). Moreover, the attacker can simulate an outside view to shadow the honest node without that node noticing that it is under attack.

To launch eclipse attacks, an attacker may use a Sybil attack to create a large number of distinct node IDs to populate the neighbor nodes of a correct node. Without controlling a large fraction of nodes, attackers may target a particular victim by forging node IDs which are close to that of the victim or launching incorrect routing updates to control most entries in the victim's routing table and neighbor list. In other words, a small number of malicious nodes colluding to control all inbound and outbound links of an honest node can launch eclipse attacks successfully. Thus, the effective defense against a Sybil attack may not be able to prevent an Eclipse attack.

Singh *et al.* [115] [117] proposed a defense against the Eclipse attack by establishing a bound on the degree, i.e., both inbound and outbound, of nodes in an overlay network and an assumption that a node that has more links than the average may be mounting an Eclipse attack. To implement this defense, each node must maintain a list of other nodes that point back to it, named a *backpointer set*. Then each node periodically uses an anonymous challenge mechanism to audit neighbor nodes to ensure the compliance with the degree bound. Moreover, to ensure that responses are fresh and authentic, the auditee includes a nonce and digitally signs the response. This **digital signature** ensures that the auditee is not being impersonated and that the response has not been modified.

2.6.5 False Claim Resource ID Attacks

To retrieve a resource by means of a key, an overlay network routes the retrieving message to the key's root node which is the node ID closest to the key. This retrieving operation relies on intermediate nodes along the routing path because any intermediate nodes can respond and claim that it owns the key's resource or that the resource does not exist. For storing a resource, a malicious node on the routing path may discard the resource, while replying with a successful response. The impact of this attack includes denial of service,

fake responses, or data modification. This attack is called **Identity Theft** because an attacker steals the node ID (identity) of a true root node.

Ganesh and Zhao [43] proposed a solution in which each node periodically signs *proof-of-life certificates* (which are used to prove the existence of the node) and distributes them to the random proof managers in the P2P network. When a client stores or retrieves data, the client can verify that a node claims to be the key's root node by requesting the proof for the existence of the node whose ID is closer to the key than the ID of the claimed root node. For the example of Tapestry (see Appendix A.5) with L-digit address space, a unique public/private key pair is assigned for each prefix group, which is a group of nodes that share the same prefix address of length $l < L$. For example, a node $ABCD$ would receive public/private key pairs for itself, $ABCD$, and for all its prefix groups, A, AB, ABC. Firstly, each node signs each prefix's existence proof with the private key of the prefix group. Second, the node $ABCD$ publishes each prefix's existence proof to a set of proof managers corresponding to the prefix. The prefix's proof managers are nodes whose IDs are derived from the SHA-1 hash value of the prefix value together with several salt values, e.g., 1, 2, 3. When a query node receives a response, it can use the space density from its routing table to investigate whether a responding node is suspicious. If the query node suspects the responder, the query node will request existence proofs of the matching prefixes that are longer than the matching prefixes of the responder. If a better matching designated node exists, it means that the responder is mounting an identity attack. This solution require an **off-line CA** that has to generate and distribute a public/private key pair to all peers in each unique prefix group. Mao *et al.* [79] proposed another light-workload solution in which each node must register for a public-key certificate binding to its node ID. Thus, every signed request and response can verify the sender and data integrity. Next, a client retrieves data by sending the request message via different routes. In general, the more routes it tries, the higher the likelihood that at least one route will return the correct response.

Commonly, applications in a structured P2P system employ redundancy, lookup functions (i.e., put, get,) and routing algorithms that are based on a DHT structure. Thus, threats to the DHT structure can subvert all of these P2P applications as well. From the survey on threats to structured P2P systems, most solutions for addressing or mitigating attacks on operations of a DHT structure are based on certificate-based cryptography. Hence, a Public Key Infrastructure (PKI) or a Certificate Authority (CA) is a basic infrastructure for providing long-term identities and security in a structured P2P system. Additionally, the requirements of public-key certificates for security provision occur frequently in both the application-level and the DHT structure-level of P2P applications.

2.7 Basic Concepts and Definitions of Cloud Computing

Cloud computing is a new computing paradigm based on a change from computing resource investment to pay-per-use services like other utilities, e.g., electricity and hydro, in order to convert from capital expenditure to operation cost. Therefore, this computing paradigm is being evolved, disseminated, and adopted by IT communities rapidly. However, because of the variety of technologies in a cloud, its definitions, characteristics, services, and architectures are divergent and confusing. This situation encourages discussion and consolidation as interested parties work to achieve consistent ideas about cloud computing.

To my knowledge, cloud computing can be defined from both business and engineering perspectives. For example, Gartner, a market research firm [98], considers it to be "a style of computing where massively scalable IT-related capabilities are provided 'as a service' using Internet technologies for multiple external customers."

On the other hand, the U.S.'s National Institute of Standards and Technology (NIST) [84] says that "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models", which are defined in the following subsections.

2.7.1 Cloud Characteristics

The five essential characteristics of Cloud Computing Services (CCSs) are presented as follows:

On-demand self-service: Cloud users can request and manage resource capability, e.g., computing time, storage space, and network bandwidth, etc., on demand with minimal management effort or human interaction with cloud service providers, e.g., web portals and management interfaces, etc.

Broad network access: Cloud resources are accessible via cloud networks (e.g., the Internet) by standard protocols (HTTP, REST, SOAP, SMTP, SSH, etc.) to provide broad accessibility. In addition, the clouds should support a variety of devices (e.g., laptops, PDAs, smart phones, etc.) and client models (i.e., thick and thin clients).

Resource pooling: In multi-tenant models, resources used to provide services must be pooled and dynamically assigned and reassigned to many cloud users in order to gain more economic and efficient utilization. Furthermore, the physical resources, i.e., processing, storage, network bandwidth, etc., are located and replicated in distinct geographical areas for resilient services and disaster recovery. Generally, these resource and location managements are opaque for cloud users by implementing virtuaization, such as virtual machines, virtual storages, load balancers, virtual networks (e.g., the Internet), etc. However, cloud users may be able to specify location at the course-grained levels, e.g., country, state, or data center, etc.

Rapid elasticity: In some cases, services can quickly scale up or down according to user configuration, or automatically scale to the right size of real loads. Therefore, in a user's view, cloud computing can provide unlimited resources without upfront costs for peak loads.

Measured service: Resource usage must be monitored and metered in order to control and optimize the resource utilization and stabilization. Moreover, usage reports must be accounted for in the billing system in pay-per-use business models.

In addition to the essential characteristics, Hoefer and Karagiannis [55] proposed two more groups of characteristics, i.e., common and specific, to allow more distinctions to be made between CCSs. We here briefly recap the common characteristics as follows:

License type: Most Cloud Service Providers (CSPs) use proprietary software (e.g., modified open-source or in-house development) or licensing software (i.e., commercial products). For example, Amazon EC2 [3] is the proprietary system developed from Xen technologies [129]; the Google App Engine [6] is a proprietary platform built around the open-source Python programming language [99]. Microsoft Azure [10] uses its own software. Meanwhile, small CSPs are using open-source cloud-monitoring software because of their lack of power and influence to develop and push their own software into the market.

Intended user groups: Most IaaS and PaaS models are intended to offer services for business, although they are not limited to supporting individual users, whereas SaaS offer services for both corporate and private users. Tentatively, mobile users are another user group for CSPs.

Security and privacy: When sensitive information is kept on cloud servers, encryption and authentication should be leveraged for confidentiality, integrity, and privacy. Both

encryption and authentication used in CSPs depend on the service models. For example, IaaS users have the most control, i.e., they can independently select and manage their own security methods and keys. PaaS can craft the security methods provided by CSPs, whereas SaaS users have to use CSPs' security methods. Commonly, CSPs use Hyper Text Transfer Protocol (HTTP) to establish connections with users' web browsers. Hence, Secure Socket Layer/Transport Layer Security (SSL/TLS) can enable encryption channels and server authentications. Users' credentials for authentication can be implemented by user/password, public keys, X.509 certificates, etc. For hybrid clouds, Virtual Private Networks (VPNs) are used to provide secure and authentic links between private clouds and public clouds.

Payment system: Usually the pricing model of CCSs is pay-per-use, in which the units or units per time of resources are associated with a fixed price value. Another model is dynamic or variable pricing, in which the price relies on demand-supply, such as auctions and negotiations. Some CCSs are free of charge or free for basic services.

Standardization: Currently cloud users are locked in a vendor and switching costs are high because of no defined standards for CCSs. The lock-in vendor is the biggest problem that impedes enterprises from adopting CCSs and prevents multiple CSPs from inter-operations. Currently, several organizations are attempting to create standards for inter-actions between clouds, such as the Open Cloud Consortium [91], DMTF Cloud Computing Incubator [36], and Cloud Computing Interoperability Forum [27].

Formal agreement: Commonly, the formal agreements between CSPs and cloud users are Service Level Agreements (SLAs), which defines the level of service that cloud users or customers expect from CSPs and the compensation that providers owe to the customers if the SLA cannot be satisfied.

2.7.2 Cloud Service Models

Based on the NIST definition and the survey of Rimal and Choi [106], Cloud Computing Services (CCSs) can be classified into three categories, according to the capability that they can offer to target customers.

Software as a Service (SaaS)

Already-created applications running on cloud infrastructures are directly provided to end-users through thin user agents, i.e., web browsers. In this model, cloud users cannot control applications, platforms, and infrastructure, except through limited application preferences at the user-level. For providers, this model simplifies their work in installation, patching, updating, upgrading, testing, and protecting their intellectual property. As for end-users, the main advantage of SaaS is pay-per-use service without hardware investment, software licensing and maintenance cost. Although the idea of SaaS existed before the emergence of cloud computing, there are many successful cases of SaaS in cloud computing. For example, Yahoo [130] provides web-based email. Google Apps [47] collaborative software provides web-based services: email, calendar, contact, and chat. The Google Doc [48] package provides web-based applications for documents, spreadsheets, and presentations. Box.net [19] provides on-line file sharing and backup. SmugMug [120] provides video and photo sharing over underlying Amazon S3. SalesForce [111] provides Customer Relations Management (CRM) software. SuccessFactors [123] provides Human Resources (HR) software. This service model is the most economic way of cloud computing; however, cloud users lose all control of applications, developments, and infrastructure.

Platform as a Service (PaaS)

Application development tools and runtime environments are provided as services for software developers. The platforms typically include development tools, middle ware, database systems, programing languages and APIs. Hence, customers develop, deploy, and manage their own applications, which rely on the platform from the providers. In this way, customers pay as they go with the development platform, instead of having to license, update/upgrade and maintain development tools and runtime software. Furthermore, developers benefit from the cloud programming environment (e.g., automatic scaling, load balancing) and cloud development tools (e.g., Hadoop [51], map reduce [81], Pig [97]). For well-known examples, the Google App Engine [6] supports Java and Python frameworks. Microsoft Azure [10] supports C#, .Net library, IIS, and SQL server. SalesForce [111] offers its own proprietary language, Apex, allowing companies to develop Customer Relationship Management (CRM) applications. For PaaS, organizations can control and customize applications, regardless of platform and hardware investment. In contrast, resource and cost sharing are not as efficient as with SaaS because organizations must pay for application development and development tools.

Infrastructure as a Service (IaaS)

Provisioning from providers involves fundamental resources, e.g., computing, storage, networks, etc., in virtualization models. These virtualizations allow IT managers to deploy or control their own operating systems or applications over the virtual resources, i.e., virtual machines, virtual storage, virtual networks, etc. Thus, customers rent the virtual resources and pay as they go, instead of having to invest and set up servers, software and data centers themselves. The advantage is that customers can quickly scale up and down the resources according to the demand and market. Real examples are Amazon Web Services (EC2, S3, EBS, VPC, and elastic load balancing) from Amazon [3], Cloud hosting and Hybrid hosting from GoGrid [46], Cloudservers and cloudfiles from Rackspace [102], IBM Smart Cloud [57], Oracle Public Cloud [94]. In this service model, organizations have independence to customize applications and select platforms. Meanwhile, the total expenditure is higher because organizations must invest in both development tools and application development.

2.7.3 Cloud Deployment Models

In addition to the service models, cloud computing also can be classified by usage, management and deployment as follows:

Private cloud: This cloud is exclusively used by a single organization and typically is hosted and managed by the organization's IT department. In some cases, it is hosted off premise, managed by a third party, but is still used for only one organization.

Community cloud: Similar to a private cloud, this cloud can be hosted and managed by an organization or a third party. Unlike a private cloud, it is exclusively used by several organizations in the same community that share a mission, requirements, or policies, etc.

Public cloud: Service providers solely own, host, and manage this cloud for selling services to public customers. Meanwhile, the customers benefit from sharing resources, i.e., having more flexibility and scalability as well as paying less cost and management.

Hybrid cloud: An organization may bind a private/community cloud and a public cloud together for a single purpose. This binding can take advantages from both clouds. For example, a 'cloudbursting' uses a private cloud to run a steady-state workload,

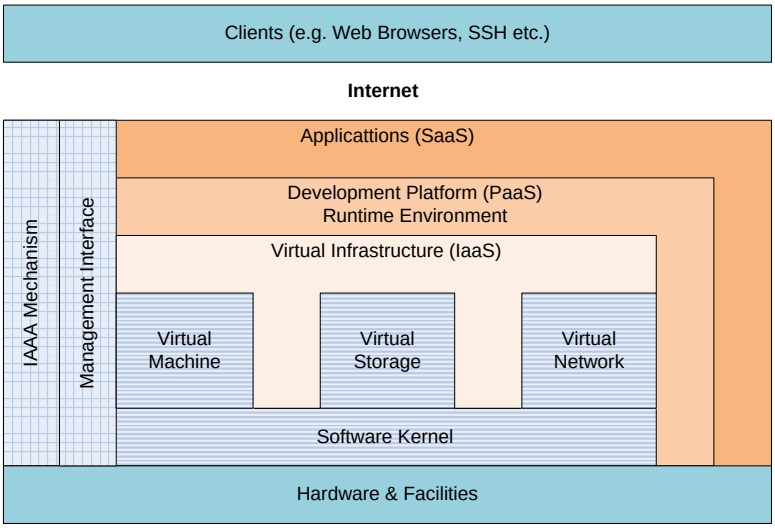

Figure 2.7: Cloud computing architecture

while moving any spikes in workload into a public cloud. To gain a security level, some providers also provide Virtual Private Networks (VPNs) to connect customers' data centers to the public cloud.

2.7.4 Cloud Architecture

In 2008, Youseff *et al.* [132] established the ontology of cloud computing by using composability methodology. We use this methodology and the NIST definition to classify cloud components as a stack of layers in two plans: service and management. The service plan consists of four horizontal layers: application, development platform, virtual infrastructure, and hardware/facilities. Each layer is composed of components that provide the same service model for the same target user group. By the order of a stack from the highest layer (application layer) to the lowest layer (hardware/facilities layer), the higher layer composes the services from the underlying layers and so on. Vertically, the management plan consists of two layers: the IAAA mechanism and management interface. We add a client layer and an Internet layer to show the untrusted network between end-users and CSPs. Figure 2.7 shows the relation among layers. In the following, the components, technologies, services, and target users of a cloud stack as well as the relation among cloud layers are explained.

Client layer: The client layer is composed of hardware and software for client devices that are used to access services deployed over clouds. The hardware can be laptops, PDAs, smart phones, thin clients, or any constrained devices. The software includes Web browsers, Secure Shells (SSHs), remote desktops, etc. Thus, there are no accessibility limits in cloud computing.

Application layer: This layer is the key to delivering the SaaS model based on web applications and services. Application software on the server side leverages cloud computing to take almost all the work load of applications, thereby exporting computation cost from the client side to CSPs. This situation also alleviates the burden of software maintenance and installation in the client side by moving all IT workload to CSPs. Hence, this layer provides already-created software services to end users.

Development platform layer: In this layer, the application development framework that consists of languages, APIs (e.g., Java, Python, Ruby, etc.) inter-operations services (e.g., Web services, database services, queue services, etc.) is provided for facilitating and accelerating application development. This layer also provides a runtime environment for deploying the applications developed from the platform. Clearly, the users of this layer are application developer, and PaaS is the name coined for the services delivered from this layer.

Virtual infrastructure layer: With virtualization technologies, this cloud layer directly provides virtual resources as a service called IaaS to IT departments/individuals, or supports other CSPs to establish the upper cloud layers (i.e., software platform and application) for PaaS and SaaS services delivery. However, the upper layers may bypass this layer and then directly build their services atop the hardware/facilities layer. Although this bypass can enhance efficiency, it increases the development cost. The virtual infrastructure layer consists of four sublayers:

- Virtual machine sublayer. Virtual machines (VMs) are the most common technology for providing computational resources in virtualization environments. With virtual machines, users have flexibilities to configure and customize their own VMs because users get full privilege to do so. Currently, hardware technologies, such as multi-core CPUs and advance chipsets are designed for hardware-assisted virtualization in order to enhance efficiency, for example, Intel VT-x processors and Intel VT-d/VT-c chipsets [59].

- Virtual storage sublayer. Virtual storage is data storage that allows users to remotely store and access their data anytime from any place (e.g., from any on-premise sites, CSPs, or Internet-connected devices). These systems are expected to support rigorous requirements for maintaining user data, for example, reliability, replication, consistency, confidentiality, integrity, privacy, etc.
- Virtual network sublayer. Because of the necessity of Internet connections with users and among virtual resources, cloud users also need communication capability that is flexible for configuration, scheduling, Quality of Service (QoS), and security. To this end, CSPs must provide virtual networks that can support traffic isolation, dedicated bandwidth, encryption channels, network monitoring, Virtual Private Networks (VPNs), etc.
- Software kernel. This sublayer facilitates the virtualization and management environment on top of the physical resource in the underlying hardware/facilities layer. Software in this sublayer is composed of OS kernel, hypervisor, virtual machine monitor, clustering middle-ware, P2P middle-ware, etc.

Hardware/facilities layer: The bottom layer of the cloud stack consists of physical hardware and facilities that form the data center for cloud service providers. This hardware includes severs, physical storage, switches, load balancers, firewalls, etc. Moreover, operating the hardware requires support from other facilities: on-premise space, shelters, electricity supply, cooling systems, and Internet connections.

Management interface layer: The rapid elasticity characteristic of cloud computing necessitates this interface to allow cloud users to control and manage instance or application deployment. This layer also facilitates cloud users' ability to rapidly scale up and down the cloud resources according to demand.

IAAA mechanisms layer: Since both service plans and management plans have to be protected from unauthorized access, CSPs must provide an Identity management, Authentication, Authorization, and Auditing (IAAA) layer as the fundamental service to facilitate the access control mechanisms in both plans. CSPs that support IAAA and identity federation for customers can extend and accelerate migration of traditional applications from local data centers to CSPs. On the other hand, cloud users can use IAAA facilities to accelerate software developments, control their resource/data access, and provide inter-operations with federated entities. For example, sales and support staffs access Saleforce.com with corporate identities. CSPs allow IT administrators to access PaaS/IaaS

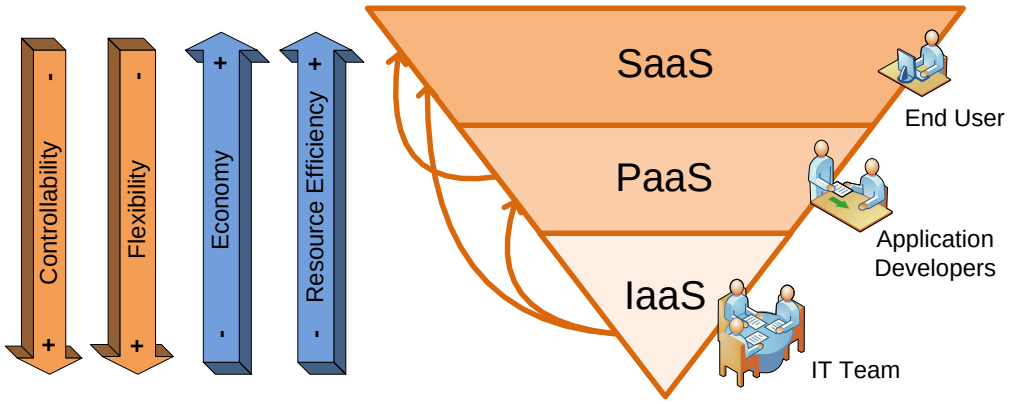

Figure 2.8: Target users and trade-off in cloud service model

management consoles with corporate identities. Developers create accounts in PaaS for part-time programmers. End-users share files/objects in Amazom S3 with other users within/outside Amazon's domain. Applications in clouds access other cloud storage.

In conclusion, cloud providers can deliver services in three models (i.e., SaaS, PaaS, and IaaS) from the cloud layers (i.e., application, development platform, and virtual infrastructure) for target users (i.e., end-users, developers, and IT teams) respectively. According to the service delivered in each layer from top to bottom, flexibility and controllability of services increase. Meanwhile efficiency and economy of resource utilization decreases. Therefore, organizations should assess these factors when selecting a suitable services from cloud computing. Figure 2.8 pictorially shows all of these relations and trade-offs.

More and more individuals and organizations are being placed in cloud computing environments, even while concerns are increasing about the security and privacy in clouds. Thus, before we accept and adopt cloud computing in our businesses or individuals, we should study and understand the security issues, adversaries, vulnerabilities and possible threats in cloud computing. The next section explains the security issues in cloud computing.

2.8 Security Issues in Cloud Computing

Cloud computing users often have more concerns about security and privacy than users of other types of distributed systems, e.g., client-server, grid computing, or P2P systems. For example, the sensitive codes and data in client-server systems are processed in under-

control servers, while untrusted clients process insensitive data or codes. Grid computing may process sensitive scientific data that are often isolated and not valuable. P2P systems, in which every node is equivalent to the others, are rarely matched to business applications that are hierarchical models. Therefore, prospective cloud users must understand and accept the following security awareness before going ahead with clouds.

A 1: Be willing to run sensitive applications and data under control of third parties.

A 2: Trust in the isolations of instances (running codes in visualized machines) and storage in multi-tenant environments managed by cloud providers.

A 3: Clearly understand a boundary of security function between cloud users and providers– how clouds provide security functions and which ones are left to cloud users.

Until now, businesses and individuals have adopted cloud computing on the basis of trustworthiness in clouds' administration like in their own; however, cloud computing still imposes challenges as follows.

C 1: How cloud users and providers cooperate to protect instances and data from external adversaries.

C 2: How cloud providers ensure isolation among the instances in the same base systems (i.e., infrastructures, platforms, or applications) to protect against internal adversaries exploiting leakages of information to launch covert or side channel attacks.

C 3: How to ensure Confidentiality, Integrity, and Availability (CIA) of services and data in clouds.

C 4: How to maintain assurance in security and privacy from clouds when providers are untrusted.

Motivation from the previous awareness and challenges leads to the next section, which discusses the adversaries in cloud environments.

2.9 Adversaries in Cloud Computing

Adversaries in clouds are classified according to adversary locations because damage levels depend on where the adversaries are placed. Thus, we classify the threats into three levels as follows.

External threats: As with clients and local servers, if services in clouds are exposed to the Internet, the services will face traditional attacks from adversaries outside the cloud's premise. For instance, adversaries may launch network attacks, such as spoofing, eavesdropping, Man In The Middle (MITM), Denial of Service (DoS), etc. Additionally, adversaries may mount malware, such as viruses, worms, Trojan horses, rootkits, spyware, etc., on instances/data in clouds or images in marketplaces.

Internal threats: Similarly, internal adversaries, who are also members of the same cloud services, can launch the same types of attacks as external adversaries can. In addition to these attacks, if internal adversaries can intentionally deploy instances or access data in the same physical resources (i.e., CPU cores, disks or I/O channels) as the ones in which victims do, the adversaries may exploit multi-tenant environments to launch other attacks, such as covert or side channel attacks. Moreover, adversaries can gain knowledge of a target's systems and platforms that are shared in the same CSP.

Provider threats: If CSPs are not trustworthy, they can take charge of users' data and processes and manipulate them as they own assets. For instance, the providers may modify, fabricate, eavesdrop on, the data or processes. Additionally, the providers may perform unauthorized data analysis (e.g., data mining), or the data may be locked into proprietary formats of the providers. These threats inhibit organizations and individuals from adopting cloud computing. Therefore, to promote more widespread use of cloud computing in the future, new security innovations to address these threats are necessary.

Generally, external threats against cloud computing can cope with current security tools (e.g., X509 certificate and SSL/TSL). Thus, we emphasize only specific cloud vulnerabilities in the next section.

2.10 Cloud Computing Vulnerabilities

This section recaps the specific cloud computing vulnerabilities from the survey of Grobauer *et al.* [50]. The vulnerabilities are divided into two groups according to the root causes: core technology and state-of-the-art of cloud computing. Before presenting the cloud vulnerabilities, we first briefly recall the definition of vulnerability from ISO 27005 [60]: "Vulnerability is the probability that an asset will be unable to resist the actions of a threat agent. Vulnerability exists when there is a different between the force being applied by the threat agent, and an object's ability to resist that force."

To provide IT services on an economic scale, cloud computing combines many technologies in the new setting. Unfortunately, each technology has vulnerabilities that are intrinsic to itself or prevalent in its state-of-the-art, as follows.

Web applications and services: SaaS typically is implemented as Web applications, while PaaS commonly provides a development platform and runtime environment for web applications and services. Commonly, IaaS provides management interfaces through Web portals. To sum up, all cloud service models absolutely rely on Web technologies. Unfortunately, Web technologies are based on the HTTP protocol, which is a stateless protocol, whereas Web applications have to maintain the state of sessions. Thus, many session-handling techniques have been developed to solve this problem; however, some techniques suffer from session riding/hijacking. Furthermore, Web developers tentatively increase using techniques based on browser-based computing, such as Java script, Java, Flash, and Silverlight. Therefore, Web application security also relies on browser components running within a user's browsers, and the isolation mechanism of Web browsers.

Virtualization: Primarily virtualization techniques are the heart for establishing virtual environments in IaaS. In addition, both SaaS and PaaS are usually built on top of IaaS infrastructures or based on their own techniques to establish virtualization. Thus, virtualization is technology crucial to cloud computing; however, this technology comes with the following intrinsic vulnerabilities:

- Virtual isolation: Attackers may successfully escape from a virtualized environment. For example, attackers may launch side-channel attacks against the virtual machines sitting in the same physical resource. Virtual storage may be prone to cross-tenant data access. Adversaries may eavesdrop on information that leaks from the shared communication.
- Image sanity: virtual machine templates (called images) cause OS and platform vulnerabilities because attackers can analyze configurations, patches, and codes in the image by renting virtual machines from the IaaS. Moreover, some IaaSs, such as Amazon EC2, allow cloud users to get images from the marketplace of virtual images. These images may be illegal in terms of software licenses, or malicious in that they bundle of malware into the image or provide a back-door for attacks.

IAAA: All cloud-service and cloud-management interfaces, including authorized end-user accesses, must be protected from unauthorized access and attackers by using the mechanisms from Identity management, Authentication, Authorization, and Auditing (IAAA). Hence, all vulnerabilities associated with IAAA must be regarded as cloud vulnerabilities. For example,

- Weak authentication credentials: Login/password authentication may be weak because users choose weak passwords, use the same passwords for many authenticators, or reveal passwords when delegating authorization. One-factor authentications have inherent limitations. Authentication protocols may be intercepted and replayed as user credentials. Unfortunately, web access controls employ these weak authentication methods, thereby incurring cloud computing vulnerabilities.

- Denial of service by account lockout: Attackers can exploit security controls that lock out accounts having received several consecutive unsuccessful login processes to disable any user accounts.

- Weak credential-reset mechanisms: If CSPs manage theirs own authentication system rather than use federal authentications, such as OpenID, etc., then the CSPs must provide their own credential-reset mechanisms, which are used when users forget or lose their credentials. The weak credential-reset mechanisms may be exploited to steal accounts.

- Coarse authorization control: Cloud management interfaces require authorization control according to the user duties in order to provide users with only the privileges they are supposed to have. Currently, most CSPs only provide coarse-grained access control.

Cryptography: Presently, cryptography techniques are the main technologies that can meet the security requirements of cloud computing. Since new cryptanalysis advances can break some encryptions and forge some signatures, cloud computing must be aware of insecure or obsolete cryptography algorithms. Some of these algorithms are based on the random numbers from the OS kernel or hardware shared in multi-tenant models. With this situation, attackers may trap the random number, or exhaust the entropy of the random number by exploiting multiple virtual machines in the same host.

Meanwhile, all the above cloud vulnerabilities are based on cloud users trusting in their CSPs. Another cloud vulnerability is cloud service providers themselves because users

lose control of their own IT infrastructure and data. Therefore, **untrusted-provider** vulnerability is the greatest challenge that impedes the adoption of cloud computing. Next, we present some attacks that exploit the above vulnerabilities to attack cloud computing.

2.11 Threats to Cloud Computing

Although many types of attacks are possible in cloud computing, some are also common attacks in local servers. Hence, this section describes only specific attacks to cloud computing.

2.11.1 Web Interface Attacks

A Web interface is a common tool that clouds use to provide services, i.e., SaaS, PaaS, and IaaS. For example, both thick and thin clients use Web browsers to access applications. For PaaS services, developers develop application code based on Web APIs (e.g., HTTP, SOAP, XML, etc.). Most IaaS services provide Web portals for storage access and instance deployment. Therefore, security in clouds relies on Web technologies. Jensen et al. [63] summarize the major flaws and solutions of Web technologies that affect cloud computing as follows.

XML signature element wrapping: Using any security flaws in XML signatures, adversaries exploit wrapping technique to hide the original SOAP body and insert a faked body with a bogus request in eavesdropped packets, so the adversaries can deploy any arbitrary requests on behalf of legitimate users. For example, Amazon EC2 was vulnerable to this attack.

Legacy same origin policies: Typically, script languages for Web browsers (e.g., JavaScript, VBScript, etc.) get access rights to read/write contents from the same origin as the scripts do. This policy is named a 'Same Origin Policy', and the origin is defined in the context of the tuple $(domainname, protocol, port)$. Unfortunately, Domain Name System (DNS) caches can easily be poisoned, and so such policies can be exploited by attackers.

Unsecured browser authentication: Commonly, Web browsers lack cryptographic credentials for authentication, and so use login/password authentication. If application servers do not maintain a login database themselves, they must redirect authentication requests to an authentication server. The authentication server generates a token associated

with a login/password entered by the user, and then redirects the token to the application servers. Due to a lack of binding between the browser and the token, adversaries may eavesdrop on the token and use it to access all services on behalf of legitimate users. For example, Windows Azure uses Passport server and REST protocol to authenticate and deliver tokens.

Although based on the Transport Layer Security (TLS) protocol, there are many ad-hoc solutions, such as TLS Federation, SAML 2.0 Holder-of-key Assertion Profile, Strong Locked Same Origin Policy, and TLS session binding. The final solution to cope with Web interface attacks is the security improvement in both Web browsers and Web service frameworks, such as extending Web browser with XML encryption and signature APIs.

2.11.2 Illegal or Malicious Image Attacks

Another cloud responsibility is to maintain the security and integrity of cloud images (i.e., Virtual Machine (VM) images for IaaS and implementation modules for PaaS). because cloud images determine the initial running and security states of the running machines and applications. In other words, cloud images' security and integrity form the foundation of cloud security. Wei *et al.* [128] described the security risks from illegal or malicious image attacks according to the parties' roles, as follows.

Publisher's risk: Publishers may release sensitive information unintentionally; for example, configuration, password or history files may be published with the image. Moreover, publishers may want to share her images with a limited group of users.

Retriever's risk: Running malicious images is equal to bringing adversaries into a protected zone, thereby bypassing any firewalls. In addition, because of running the malicious images, some malware may be mounted on the victims' VMs . Furthermore, retrievers also take the risk of running illegal software, i.e., unlicensed or expired-license software.

Repository administrator's risk: Images in a repository are dormant but not static. That is, published images are initially secure but exploitable later. Software licenses is valid initially but is expired later. Therefore, repository administrators risk hosting and distributing malicious or illegal images.

Unfortunately, the access control and encryption techniques for cloud storages are not adequate for cloud images. For instance, VM images are not static objects that include their initial states (in the case of a snapshot), patches, and transformation. To cope with

the previously mentioned risks, Wei also proposed an image management system, called Mirage, which provides the following features: 1) an access control that regulates the sharing of VM images; 2) image filters that are applied to an image when publishing and retrieving images in order to remove unwanted information in the images; 3) a provenance tracking mechanism that tracks the histories of derivation, modification, and operation that are applied to images, and also provides accountability and auditing; 4) repository maintenance services, such as compliance checking, virus scanning, patching, etc.

2.11.3 Cache Interference Attacks

In multi-tenant environments, multiple instances of Virtual Machines share the same physical resources, so it is the responsibility of cloud providers to isolate an instance from other instances in terms of both performance and security. Typically, virtualization is used to encapsulate instances inside virtual machines for virtual isolation, ease of service deployment, and flexible migration. However, virtual isolation is weaker than physical isolation because some physical resources, such as Last Level Caches (LLCs) and memory bandwidth, in the multi-core environments are implicitly shared among instances. These implicit shares present opportunities for security or performance interference. Due to the prevalence of multi-core environments in clouds, malicious instances can exploit cache interference to launch Dos, side/covert channel attacks on other instances. To cope with these cache interferences, a hypervisor (i.e., a VM manager in a virtualization system) may use resource (e.g., memory/cache) management to isolate memory/cache used by each instance. Two well-known techniques for coping with cache interferences in cloud computing are summarized by J. Wei, et al. [128] as follows.

Cache-hierarchy-aware core assignment: Currently multi-core systems are configured with a multiple-package structure in which all cores share the same LLC. This structure provides cache isolation between each package thereby presenting an opportunity to exclude cache sharing. In this technique, if services require cache isolation, the hypervisor tries to choose a core in a package currently assigned to the same instance or an empty package. Without cache isolation requirement, any cores in any shared packages can be assigned. Although this technique is easy to implement, it causes under utilization in cloud infrastructures.

Page-coloring-based cache partitioning: Typically, most Operating Systems use paging techniques for virtual memory. Hence, the hypervior can leverage a page-coloring technique, which is a software method of mapping between physical memory and cache

hardware. Using page-coloring technique, both physical memory and cache hardware are partitioned into multiple groups called colors. In virtualization, a hypervisor, who takes control of memory paging and cache paging for VM, can utilize page coloring for cache isolation. If services require cache isolation, the hypervisor assigns a distinct set of colors to individual instances in the same shared LLC (i.e., belonging to the same package), otherwise any set of colors. This technique has an advantage over the former technique in that it does not cause under utilization of cores. However, some memory in a color page may not be used if an amount of required memory is not multiples of the color-page size.

2.11.4 A New Form of Denial of Service Attacks

Generally, network resource in a data center are grossly under-provisioned. For local data centers, the typical network designs are under-provisioned by a factor of 2.5:1 to 8:1 (meaning that network-interfaces can support only 1/8 of the maximum or some patterns of communication loads). Since IT teams have the full control of infrastructures, this under-provisioning problems can be mitigated or addressed by redesigning network infrastructures, tracking down offending applications, and putting countermeasures in the systems. Unlike in local data centers, the under-provisioning in cloud computing lead to a new form of Denial of Service (DoS) attacks because 1) the under-provisioning in clouds is dramatic, at a factor of 45:1 or more, 2) cloud data centers are shared by multiple organizations thereby presenting opportunities for adversaries to attack others applications in the same cloud data center, 3) application owners have limited or no control over the underlying networks in cloud data centers.

H. Lui [77] presents a feasible DoS attack that exploits the gross under-provisioning of routers' up-link bandwidth. To launch this attack, 1) adversaries launch a set of instances in cloud data centers. 2) Then these adversaries can learn the network topologies of clouds by using a common network tool, 'Traceroute', or exploiting probing techniques based on the multiplexing nature of routers to learn clouds' network topologies. 3) For accuracy of learning topologies, adversaries may use general network capacity estimation tools to estimate the required number of instances for learning, and exploit the knowledge about instance assignment algorithms to optimize the required number of instances. 4) Once adversaries have a large enough number of appropriate instances and have knowledge about network topologies, they can send traffic (UDP packets) through upstream routers to saturate the target router's up-link, thereby launching a new form of DoS attacks, i.e., by disabling router up-links. Since the adversaries do not send traffic to the target application, so traditional DoS or Distributed DoS (DDoS) countermeasure tools cannot track this new from of DoS. Lui also proposed a DoS avoidance mechanism using monitor agents on the

premises of the service owners or other CSPs to monitor bandwidth starvation. Then the monitor agents will activate standby instances or launch new instances in different subnets or clouds if the main instances are attacked.

2.11.5 Miscellaneous Attacks

Injection: This type of attack manipulates services or applications by inputing commands or scripts such that parts of them are interpreted or executed in the wrong way, i.e., against the programmers' intentions. For example, SQL injection causes erroneous executions in back-end databases, command injection causes erroneous executions via OS, cross-site scripting is Java Scripts erroneously executed by Web browsers, etc.

Media sanitization: In addition to data recovery, media sanitization, such as data destruction policies at the end-of-life cycle, is hard or impossible to implement in the cloud environment. For example, physical media cannot be destroyed if other tenants still use those media. While cryptography is usually used to encrypt data in those media, cloud users must be aware of obsolete cryptographic tools and poor key management systems.

Chapter 3

Security Models and Definitions

3.1 Security Associations for Peer-to-Peer Systems

3.1.1 Public-Key Infrastructures

Generally, security functions are based on trust relationships among different entities. However, trust relations cannot be established from thin air and instead depend on existing relationships. Therefore, a security function requires an infrastructure to support trust relationships. This infrastructure can provide an on-line or off-line service according to whether or not an entity needs to interact with the infrastructure when a security function is executed. For certificate-based cryptography, a certificate, which is signed with the private key of a Certificate Authority (CA), binds a public key with the identity of its owner. In this scenario, if every entity trusts the CA and recognizes the CA's public key, a public-key certificate is a crucial tool for converting a multiple peers-to-CA trust relation to an individual peer-to-peer trust relation [24]. In addition to an identifier and a public key, the certificate is comprised of a limited lifetime, an issuer (the CA's name) and other information, all of which are indicated in signed contents. Because users can individually verify these contents with a CA's public key, a certificate can be distributed via an unsecured communication channel and cached in untrusted storage. To achieve this functionality, the simplified architecture of Public Key Infrastructure (PKI), shown in Figure 3.1, consists of the following parties:

End entity: A user of a PKI certificate or subject of a certificate,

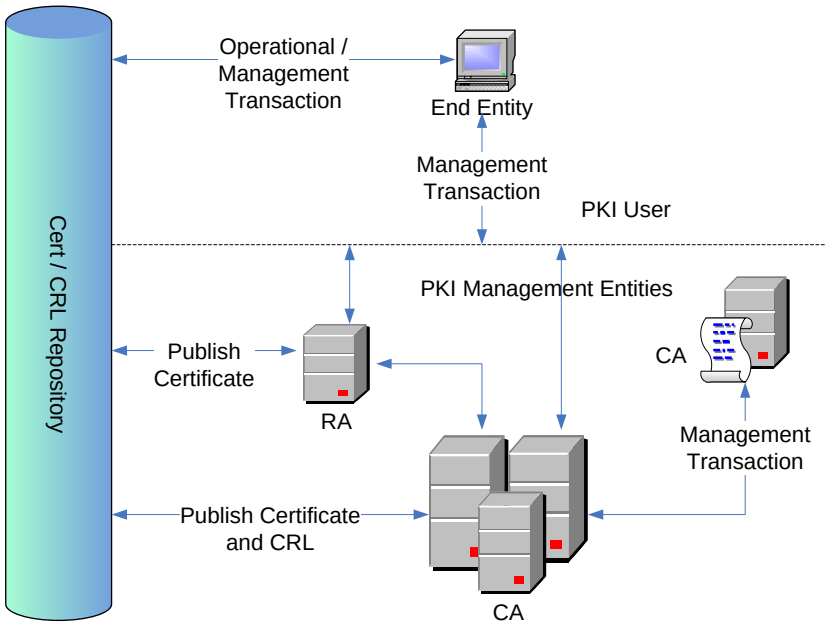

Figure 3.1: The architecture model of PKI

CA: A certificate authority,

RA: A registration authority (optional),

Repository: A system that stores certificates or CRLs for serving to end entities.

To obtain a certificate, an end entity must generate a public/private key pair herself and keeps the private key secret. Then she generates a certificate request message based on the PCK 10 standard [88] from her identifier and private key and signs the certificate request with her private key before sending the certificate request to a CA. The CA validates the request by verifying her identifier and public key according to the CA's policies. If the verification is valid, the CA creates a public-key certificate associated with the certificate request, then sends it back to the requester.

After a certificate is issued, it is expected to be used for its entire lifetime. However, many circumstances can cause a certificate to be revoked prior to the expiry time, for example, change of name, change of association between subject and CA (e.g., change of a job position or employment termination), and key compromise. The X.509 standard defines a method for certificate revocation by which a CA issues a signed data structure

called a Certificate Revocation List (CRL) and publishes it in a public repository. To verify the validity of a certificate, an end entity not only checks the validity time of a certificate but also queries the certificate ID in the CRL.

The data structure and procedure for a certificate and a CRL are defined by the X.509 standard, which uses ASN.1 syntax and Distinguished Encoding Rule (DER) encoding. This standard was first published in 1988, then revised in 1993 and 1996, (Versions 1, 2 and 3, respectively). The main extensions were developed in Version 3 [56], which adds information for subject identification, key attributes, key/usage policies, certificate path constraints and private usage. Similar to a certificate, a CRL can be distributed via an unsecured communication channel and stored on an untrusted server. However, a CRL has a time delay, e.g., one hour, one day, or one week, which counts the time from when a certificate revocation is reported until the next periodic CRL is issued.

3.1.2 Byzantine Agreement (BA) Algorithm

In our thesis, the main purpose of an agreement on digitally signed messages is to detect a malicious node in a decision group. Therefore, we must guarantee that every honest node reaches the same decision so that the majority of nodes in a decision group can reject a malicious node in the system. Before we describe this detection algorithm in Section 4.4.3, we present the original idea of a *Byzantine Agreement Algorithm*.

In general, a reliable system is implemented by using several different input components (input nodes) to compute the same result and then perform a decision function, e.g., *majority, maximum, median, minimum,* etc., on their results to obtain a single value. If every decision node obtains consistent input(s), the non-faulty node will produce the same output. (Note that the decision may be performed within the system, so a node in the system may function as both an input node and a decision node). However, the source of a single data may come from a faulty node (e.g., a faulty processor, a faulty sensor, etc.) that gives different values to different nodes. Therefore, an agreement algorithm requires the following conditions to hold for every non-faulty node [45]:

- All non-faulty nodes must use the same input vector, $\overrightarrow{V} = (v_1, v_2, ..., v_n)$, where v_i is the input value from the i^{th} process.
- If the input unit (i^{th} node) is non-faulty, then all non-faulty nodes use the value $V[i]$ of their vector as the input value of the i^{th} process.

These are the *interactive consistency* conditions whose goal is to agree on the vector of a value proposed by n nodes $P_1, P_2, ..., P_n$.

In 1982, Lamport *et al.* [71] proposed the idea that the *Interactive Consistency (IC) problem* can be reduced to a general problem, called the *Byzantine General (BG) Problem*, in which only one node (a *commander*) proposes its value in the system, and the other nodes (*lieutenants*) try to reach an agreement on the commander's value. In other words, this BG problem can be extended to an IC problem by running BG n times, once for each node acting as a commander. As a result the solution and boundary of BG problems can apply to the original IC problem. Moreover, the authors proposed two algorithms to cope with the BG problem: *Oral Message* and *Digitally Signed Message* solutions. The definition of an oral message (or a digital unsigned message) is based on the following assumptions:

A1: Every message sent by a non-faulty node is delivered correctly by a direct communication between a sender and a receiver. Therefore, a non-faulty node cannot be interfered with in the communication.

A2: A node can determine the originator of any message it receives, meaning that no faulty nodes can impersonate a non-faulty node.

A3: The absence of a message as well as an omission failure can be detected. The solution for this assumption is a time-out convention that requires two more assumptions:

- There is a fixed maximum processing and transmission time for a message.
- The time drift between sender and receiver clocks has a lower bound.

Based on assumptions A1, A2 and A3, if the number of nodes $n \geq 3f + 1$, (where f is the number of faulty nodes), then a BG problem can be solved with the lower bound of $f + 1$ rounds and $O(n^{f+1})$ oral messages. For a digitally signed message solution, if all messages are signed, we can omit assumption A2, but we must add another assumption A4:

A4: A non-faulty node is able to sign its messages in such a way that no faulty nodes can forge its signature, and any other nodes can verify the authenticity of this signature. Note that the signatures of faulty nodes can be forged by other faulty nodes, thereby launching a collusion attack on the system.

Table 3.1: The BA matrix of a message with digital unsigned messages

Round		P_1 received	P_2 received	P_3 received	P_4 received
1^{st}	P_i sent	$[1, 2, w, 4]$	$[1, 2, x, 4]$	$[1, 2, y, 4]$	$[1, 2, z, 4]$
2^{nd}	P_1 sent	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$
	P_2 sent	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$
	P_3 sent	$[a, b, c, d]$	$[e, f, g, h]$	$[i, j, k, l]$	$[m, n, o, p]$
	P_4 sent	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$
$majority(\cdot)$		$[1, 2, \perp, 4]$	$[1, 2, \perp, 4]$		$[1, 2, \perp, 4]$

Based on assumptions A1, A3 and A4, if the number of nodes $n \geq f + 1$, then a BG problem can be solved with a lower bound of $f + 1$ rounds and $O(n^{f+1})$ digitally signed messages.

To illustrate how the solution for BG problems works, we use a small group of four nodes P_1, P_2, P_3, and P_4 that propose values of 1, 2, 3, and 4, respectively. Node P_3 is a faulty node that lies consistently. Additionally, a simple algorithm consists of two rounds and one decision step:

Round-1: Each node sends its own value to other nodes;

Round-2: Each node forwards received values to others;

Decision: Each node locally votes on each element of its own vectors by a majority function.

Table 3.1 shows the matrices of information that each node received in the first and second round, and the output of a majority function from the oral message algorithm. Table 3.2 shows the matrices and the output from the digitally signed message algorithm. Note that we denote $\perp$ as an undefined value, and for a digitally signed message solution, a sender signs every message. As one can see in the matrix of both oral and digitally signed message solutions, the local output of each node has an agreement, but the value of node P_3 is an undefined value. Therefore, we can detect that P_3 is a faulty node if P_3 lies consistently until a majority of the received values are corrupt.

Some techniques can be used to optimize the number of rounds and messages of the Byzantine agreement algorithm. For example, in 1983, Dolev and Strong [37] proposed

Table 3.2: The BA matrix of message with digitally signed messages

Round		P_1 received	P_2 received	P_3 received	P_4 received
1^{st}	P_i sent	$[1, 2, w, 4]$	$[1, 2, x, 4]$	$[1, 2, y, 4]$	$[1, 2, z, 4]$
2^{nd}	P_1 sent	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$
	P_2 sent	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$
	P_3 sent	$[1, 2, a, 4]$	$[1, 2, b, 4]$	$[1, 2, c, 4]$	$[1, 2, d, 4]$
	P_4 sent	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$
$majority(\cdot)$		$[1, 2, \perp, 4]$	$[1, 2, \perp, 4]$		$[1, 2, \perp, 4]$

a variant of BG using digitally signed messages, based on eliminating duplicated signed messages that a non-faulty node is not able to forge. In 1983, Rabin [101] proposed a Randomized Byzantine General Algorithm for a probabilistic solution in which a number of messages and rounds is lower than the lower bound of deterministic algorithms. Although the complexity of the variants of the Byzantine agreement algorithm is less than that of Lamport's Byzantine algorithm, these variants come with reduced accuracy as well as reduced redundancy.

3.1.3 Threshold Signature

In 1979, Shamir [114] proposed a (t, n) threshold scheme that is suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate in order to protect a key, d. In this scheme, d is divided into n pieces, i.e., $d_1, \ldots, d_n$, such that knowledge of any t or more pieces of d_i makes d easily computable, but knowledge of any $t-1$ or fewer pieces of d_i leaves d completely undetermined. To achieve these properties, this scheme uses a polynomial interpolation (i.e., Lagrange interpolation):

- t distinct points $(x_1, y_1), \ldots, (x_t, y_t)$, where there is only one polynomial $f(x)$ of degree $t-1$ such that $y_i = f(x_i)$ for all i
- $f(x) = \sum_{i=1}^{t} f(x_i) \cdot l_i(x)$, where the Lagrange polynomial at x,

$$l_i(x) = \prod_{j=1, j \neq i}^{t} (x - x_j)(x_i - x_j)^{-1}$$

and at $x = 0$

$$l_i(0) = \prod_{j=1, j\neq i}^{t} \frac{x_j}{x_j - x_i}$$

From [31], a dealer distributes n shares of key d among n nodes (shareholders) $P_1, \ldots, P_n$ as follows:

- Choose a prime $p > \max(d, n)$.
- Pick a degree $t-1$ polynomial $f(x) = a_0 + a_1 x + \ldots, a_{t-1}x^{t-1}$ such that $a_0 = d$ and $a_1, \ldots, a_{t-1} \in \{0, \ldots, p-1\}$ at random.
- Divide d into n pieces $d_1, \ldots, d_n$, such that $d_1 = f(x_1), \ldots, d_n = f(x_n)$.
- Distribute (x_i, d_i) to the peers P_i, $i = 1, 2, \ldots, n$
- Compute the key $d = a_0 = f(0) = \sum_{i=1}^{t} f(x_i) \cdot l_i(0) \pmod{p}$ by any group of t or more shareholders.

The (t, n) threshold scheme achieves a balance between service availability and key compromise. An adversary has to break t shareholders to compromise the key and must destroy $(n-t)+1$ shareholders to breach the service. Two extreme cases are compared as follows: i) $t = 1$ is equal to a centralized solution (suffers from a single point of key compromise and failure); and ii) $t = n$ provides maximal security (the key is compromised only if all shareholder are broken) but minimal fault tolerance (the system fails from only one broken shareholder). By choosing $1 < t < n$, we can avoid both single point of key compromise and failure.

In 1990, Desmedt and Frankel [33] proposed a threshold public-key scheme based on Shamir's threshold scheme and ElGamal's cryptographic scheme [40]. In this scheme, a dealer randomly selects a primitive generater g over a finite field $\mathbb{F}_p$, picks a private key $0 < d < Q$ and publishes a public key $e = g^d \pmod{Q}$, where $Q = ord(g) = p - 1$. Then the dealer generates n shares $d_i = f(x_i)$, for $i = 1 \ldots n$, from the secret d according to Shamir's threshold scheme. After the dealer distributes all tuples (x_i, d_i) to every shareholder, the dealer can delete the secret d and polynomial $f(x)$ and disappears forever. Each shareholder secretly keeps the share and uses it to compute a partial result. To encrypt a message M, a sender selects a random number $0 < k < Q$, such that $gcd(k, Q) = 1$, computes the cipher text $C = Me^k$, and then sends a tuple (g^k, C) to a receiver. To decrypt a cipher text, the

receiver selects a coalition of t shareholders and asks for decryption by sending them g^k. Each shareholder modifies each share d_i to be a modified share (i.e., $a_i = d_i l_i(0) \pmod{p}$), computes a partial result $g_i' = g^{-ka_i}$, and then sends g_i' to the receiver (where $l_i(0)$ is the value of the Lagrange polynomial at point 0, i.e., $l_i(0) = \prod_{j=1, j\neq i}^{t} x_j (x_j - x_i)^{-1}$). The receiver computes a complete result (i.e., $g^{-dk} = \prod_{i-1}^{t} g_i'$) and computes the plaint text (i.e., $M = Cg^{-dk}$). To make this algorithm truly non-interactive between shareholders, each shareholder computes and sends a tuple (g^{kd_i}, x_i) to the receiver and the receiver computes each modified share a_i by itself. In addition, this scheme can be applied to digital signature and verification as well as other cryptographic systems based on discrete logarithms.

In 1992, Desmedt and Frankel [34] extended their scheme to the RSA cryptographic system [108], namely threshold RSA signatures. In this scheme, they use a modified RSA that replaces the Totient function $\phi(\cdot)$ by the Carmichael function $\lambda(\cdot)$ and a modified Lagrange interpolation that calculates over the integers and modulus $p'q'$ as follows:

- A dealer picks a degree $t-1$ polynomial $f(x)$ such that $f(0) = d - 1 \pmod{\lambda(n)}$.
- A dealer picks all x_i's to be odd and all $f(x_i)$'s to be even, $i = 1, 2, \ldots$.
- The interpolation is processed without knowledge of $\lambda(n)$, as follows:
  - (t, n) threshold, a shareholder set $\mathcal{A}$ with $|\mathcal{A}| = t$, and a threshold $\mathcal{B}$ with $|\mathcal{B}| = n$.
  - $f(x) = \sum_{i\in\mathcal{B}} d_i q_i(x) \pmod{2p'q'}$, where $q_i(x) = \prod_{j\in\mathcal{B}, j\in\mathcal{A}} (x_i - x_j) \prod_{j\in\mathcal{B}, j\neq i} (x - x_j)$.
  - A dealer calculates one share $d_i = \frac{f(x_i)/2}{[\prod_{j\in\mathcal{A}, j\neq i}(x_i - x_j)]/2} \pmod{p'q'}$ for each shareholder and sends it to that shareholder.
- Each shareholder calculates a modified share $a_i = d_i q_i(0)$ such that $d - 1 \equiv \sum_{i\in\mathcal{B}} a_i \pmod{\lambda(n)}$.

The modified RSA algorithm is operated as follows:

- Pick p' and q' to be primes, $p = 2p' + 1, q = 2q' + 1$, and $n = pq$;
- Compute the Carmichael function: $\lambda(n) = \lambda(pq) = \text{lcm}(2p', 2q') = 2p'q'$;
- Randomly choose a private key d such that $gcd(d, \lambda(n)) = 1$;

- Publish a public key $e \equiv d^{-1} \pmod{\lambda(n)}$;
- Generate a partial signature $S_i \equiv M^{a_i} \pmod{n}$, where M is a message;
- Reconstruct a complete signature

$$S \equiv M \prod_{i \in \mathcal{B}} S_i \equiv M M^{\sum_{i \in \mathcal{B}} a_i} \equiv M M^{d-1} \equiv M \pmod{n}.$$

In [69], Kong *et al.* proposed a threshold RSA scheme that uses the original RSA functions and the original Lagrange interpolation of Shamir's (k, n) threshold scheme. Although the sum of partial keys in the Lagrange interpolation is not equal to the corresponding private key, i.e., $\sum_{i=1}^{k} d_i l_i(0) \pmod{n} = tn + d \neq d$, each $d_i l_i(0) \pmod{n}$ is a value between 0 and $n-1$ due to the modulus arithmetic. Thus t satisfies the inequality $0 < t < k$. The authors use the upper bound and an algorithm, named the **K-bound Coalition Offsetting**, to find the correct signature that is proved by the corresponding private key. This method is simple and suitable for a small group of shareholders, such as in ad-hoc networks.

The standardization organization IEEE has already listed the threshold digital signature technology to issue a signature without using a trusted center and secret communication for the future work and research in IEEE P1363 [124].

3.2 Security Associations for Cloud Computing

3.2.1 Attribute-Based Encryption (ABE)

In 2005, Sahai and Waters [110] proposed a variant of Identity-Based Encryption (IBE) named "Fuzzy Identity-Based Encryption" that enables encryption using biometric inputs as an identity. Their work also introduced a new application called "Attribute-Based Encryption" (ABE), in which both a private key and a public key are sets labelled with a set of descriptive attributes. A user with a private-key set ω can decrypt a ciphertext encrypted with a public key ω' if and only if there is enough matching between key sets ω and ω'. In 2006, Goyal *et al.* [49] originated a fine-grained ABE called "Key-Policy ABE" (KP-ABE) in which the access policy is associated with a private key. In a KP-ABE, a ciphertext is simply encrypted with a public-key set, but a private-key set is bundled into an access structure that controls which ciphertext a private-key set can decrypt. In 2007,

Bethencourt *et al.* [14] proposed a new ABE scheme, called "Ciphertext-Policy ABE" (CP-ABE), realizing access control on encrypted data. Unlike a KP-ABE, this scheme bundles the access policy with the ciphertext. Since its development, ABE has been divided into two types: key-policy and ciphertext-policy. Regardless of where an access policy is associated with, such a policy is specified in term of access structure over a set of attributes, as detailed in the next subsection.

In our proposed authorization scheme, we will improve the security feature of the OAuth authorization standard [89],[90] using ciphertext-policy attribute based encryption in [14]. However, in order to adapt the CP-ABE to the scenario of the OAuth, we need to make some modifications on the CP-ABE, which will be introduced in Section 5.3.5 of Chapter 5.

Access Trees

First we recap the definition of a monotone access structure from [127] as follows:

Definition 1 (Access Structure [11]) *Let $\{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1,P_2,\ldots,P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.*

In this context, the role of parties is taken by attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized set of attributes. Here we restrict our attention to monotone access structures. Note that it is possible to have a non-monotone access structure, i.e., one having a negative attribute, by using the technique in [95], although it may not be efficient.

Definition 2 (Linear Secret-Sharing (LSS) Schemes [11]) *A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if there exists a matrix E with l rows and n columns, called the share generating matrix, such that the following are satisfied.*

1. *For $i = 1, \ldots, l$, we define the function ρ as a labeling function such that $\rho(i)$ corresponds to one party with the share that is a linear combination of the elements in the i-th row of E.*

2. *When we consider a column vector $v = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then $E \cdot v$ is the vector of l shares of the secret s according to Π. The share $(E \cdot v)_i$ belongs to party $\rho(i)$.*

It is shown in [11] that each linear secret-sharing scheme according to the above definition also has a linear reconstruction property, which is defined as follows: Suppose that Π is an LSS scheme for an access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorization set and $I \subset \{1, 2, \ldots, l\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that if $\{\lambda_i\}$ are valid shares of any secret s according to Π, then $\sum_{i \in I} \omega_i \lambda_i = s$. Furthermore, it is shown in [11] that these constants $\{\omega_i\}$ can be found in polynomial time in the size of the share generation matrix E.

In practice, an access policy $\mathbb{A}$ can be represented by a tree structure (called an access tree τ) for which each non-leaf node of the tree is a (t, n) threshold gate, which is described by its n child nodes and the threshold value t of a $(t-1)$-degree polynomial. Hence, each non-leaf node can represent an "AND" or "OR" gate in an access policy by using an (n, n) or $(1, n)$ threshold gate, respectively. Meanwhile, each leaf-node associates with an attribute in the access policy. Figure 3.2 shows the three primitive constructions of non-leaf nodes in a tree structure: a 1-of-3 threshold gate (aka an OR gate), a 2-of-3 threshold gate, and a 3-of-3 threshold gate (aka an AND gate).

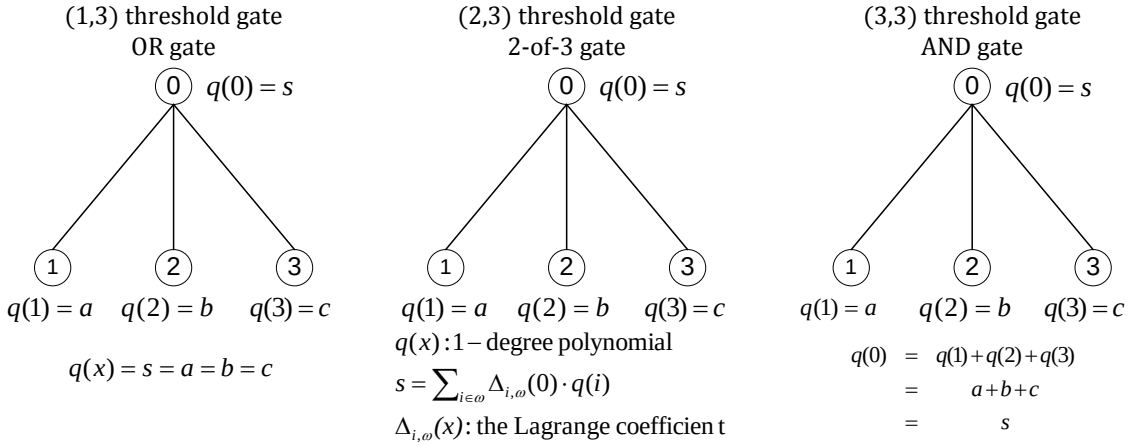

Figure 3.2: $(1, 3)$,$(2, 3)$, and $(3, 3)$ threshold gates

3.2.2 Kerberos

A Kerberos [68] consists of four roles, i.e., a client, an Authentication Server (AS), a Ticket Granting Server (TGS) and a Resource Server (RS). Kerberos's secret keys are shared and used between a client/resource server and Kerberos servers, i.e., an AS and a TGS. While a client's key is used to authenticate the client to an AS, a server's key is used to encrypt a ticket for the intended server, i.e., a TGS or RS. Moreover, a temporary session key is used for authentication and encryption in each session. The basic authentication protocol performs as follows:

1. A client sends a token request consisting of a client identity ID_{CLI}, a ticket granting server identity ID_{TGS}, and lifetime LT to AS.
2. An owner must expose his credentials to a client for decrypting a response from an AS, in order to get a session key SK_{TGS} and a Ticket Granting Ticket TGT that is encrypted with the TGS key K_{TGS} and composed of the TGS ID and the session key SK_{TGS}.
3. The client uses the TGT and the SK_{TGS} to authenticate himself to a TGS.
4. If the authentication is successful, the TGS will issue the client another session key SK_{RS} encrypted with the previous session key SK_{TGS}, and a Service Ticket ST that is encrypted with the RS key K_{RS} and composed of the RS ID and the session key SK_{RS}.
5. The client uses the ST and the SK_{RS} to authenticate himself to the RS for granting access to the objects (i.e., protected resources).
6. The RS provides resource to the client if the an authentication is successful.

To achieve a single sign-on authentication, a Kerberos issues a TGT that does not define a resource and allows a client to reuse the TGT for multiple resource requests until the TGT is expired. Figure 3.3 provides a pictorial explanation of the protocol flows in the Kerberos system.

3.2.3 OpenID Standard

To resolve the user-password authentication problem in the Internet, an open-standard authentication for consumers, named OpenID [93], provides a way to consolidate digital

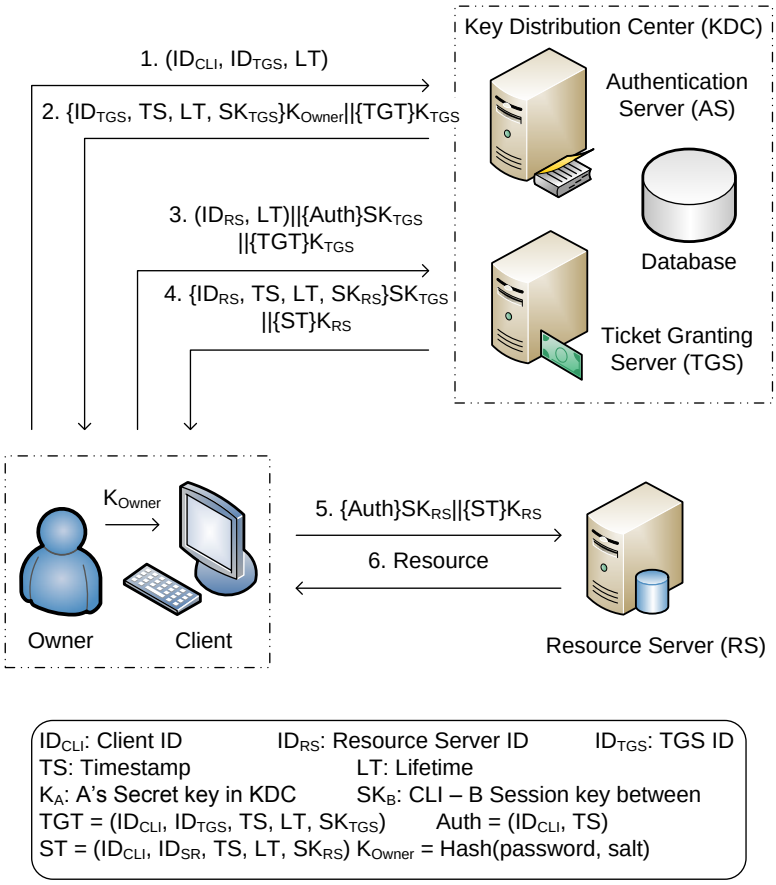

Figure 3.3: Kerberos protocol flows

IDs. Thus, web-application providers (also called the Relying Parties (RPs)) can verify a user's ID with an interactive protocol from an OpenID Provider (OIDP) without exposing a user's password to the web-application providers. In other words, an OIDP provides a single digital ID that is shared by many RPs but verified only by the OIDP. Moreover, OpenID is decentralized in the sense that no centralized authority must approve or register OIDPs or RPs, and a user freely chooses which OIDP to join and can preserve his ID if he switches to other OIDPs. Although OpenID mandates the protocol for an authentication, the standard allows the use of any authentication credentials, e.g., a password, a smart card, a client certificate, a biometric characteristic, an information card, etc. Figure 3.4 shows the protocol flows of OpenID as follows:

1. A user initiates an authentication by presenting a user-supplied ID, i.e., an ID pre-

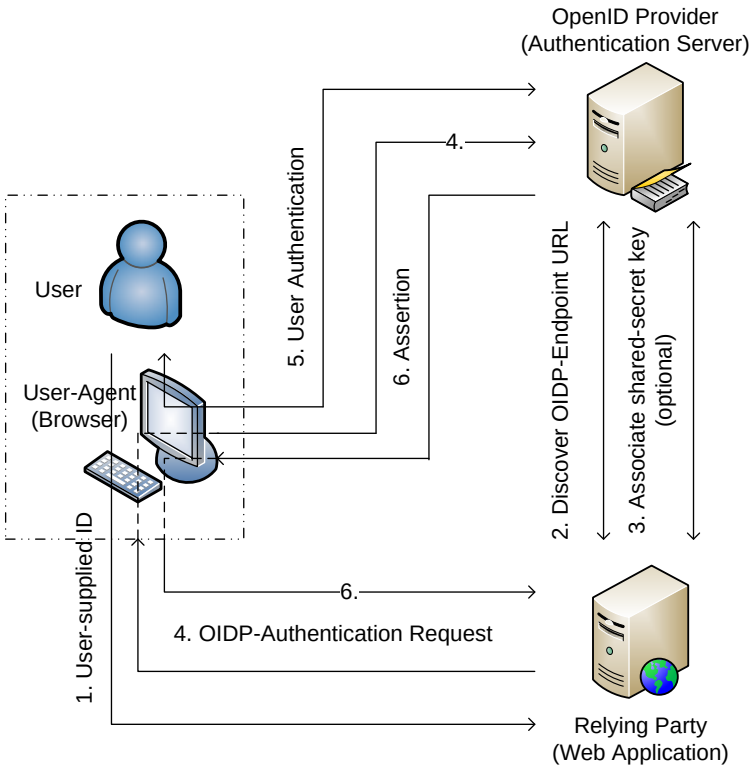

Figure 3.4: OpenID protocol flows

sented by a user to an RP.

2. The RP normalizes the user-supplied ID or a claimed ID, i.e., an ID that a user claims to own, and then discovers an OIDP-endpoint URL, i.e., a URL accepting an OpenID protocol.

3. The RP and the OIDP share a secret key, which is used for signing the message between the OIDP and the RP, by using a Diffie-Hellman key exchange (Optional).

4. The RP redirects the user's browser, termed a user-agent, to the OIDP by an authentication request.

5. The OIDP directly authenticates the user.

6. The OIDP redirects the user-agent back to the RP with an assertion, i.e., an accept or a reject. Finally, the RP verifies the assertion message with the shared key and

checks the redirection URL.

In addition, with a simple cookie and Remember Me checkbox, an OIDP can act as an SSO solution for someone who uses multiple OpenID applications.

3.2.4 OAuth Standard

An IETF working group is exploring a standard called the OAuth [89] [90] authorization protocol that allows one service provider (aka a third-party client), who is not an owner, to access the resources from another service provider (aka a resource server) on behalf of the owner without exposing the owner's secret credentials to the third-party client. To this end, the OAuth uses HTTP redirections and tokens to introduce an authorization layer, which separates an owner's role from a third-party client and provides a secure way for an owner to allow one provider to access his/her resources that are hosted by other providers. Exploiting a token not only protects an owner from sharing his credentials with a client but also limits the scope and lifetime of an access permission. In terms of technologies, OAuth is only based on the standard HTTP requests and responses, so it do not require any specific software except a common browser.

To demonstrate how OAuth works, we assume that every third-party client (called a client for short) is registered with an authorization server for client credentials (i.e., a client ID and password/shared-key) and an optional redirection URI, which are used to identify and authenticate the client itself. Figure 3.5 presents the protocol flows among its four roles, i.e., a resource-owner/user-agent, a third-party client, an authorization server (aka an OAuth Provider (OAP)), and a Resource Server (RS). The protocol consists of the following steps:

1. A client initiates the authorization by redirecting the user agent of a resource owner (called an owner for short) to an OAP, and bundles a client ID and redirection URI in the request.

2. The OAP directly authenticates the owner and obtains an authorization decision from the owner via the user agent.

3. If the owner grants access, the OAP issues an authorization code and adds it in the query that redirects the user agent back to the earlier redirection URI.

4. The client requests an access token from the OAP by using the client credentials, the authorization code, and the redirection URI from the previous steps.

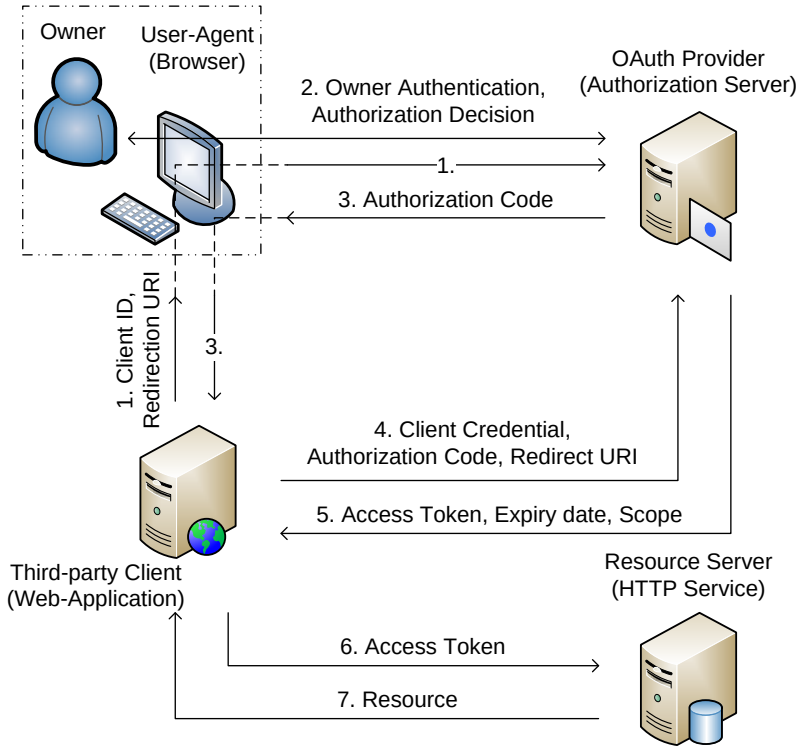

Figure 3.5: OAuth protocol flows

5. If all of the parameters in the request are valid, the OAP responds with an access token, a scope, and a lifetime.

6. The client can present the access token to the RS in order to obtain a protected resource.

7. If the access token is valid, the RS will provide the resource according to the scope.

As a result, OAuth is able to delegate an authorization grant from an owner to a client via an authorization server, in the form of a token that is used by a resource server to enforce an access policy. Exploiting a token not only protects an owner from sharing his credentials with a client but also limits the scope and lifetime of an access permission.

In terms of technologies, OpenID and OAthu are only based on standard HTTP requests and responses, so they do not require any specific software except a common browser. Moreover, to reinforce security in both standards, user agents are forbidden to use cookies

or scripts from third-party clients. Thus, a web browser and HTTP protocol function as a trust platform and act in a user-centric fashion. While OpenID focuses on authentication for HTTP-service providers, OAuth focuses on authorization for sharing resources among HTTP-service providers. However, there exists cooperation between openID and OAuth. For instance, an OAuth provider can request authentication from an OpenID provider. Note that these existing standards cannot prevent unauthorized access in some hostile environments of public clouds.

Chapter 4

A Framework Toward a Self-Organizing and Self-Healing Certificate Authority Group

A public-key certificate is an effective way to solve various security problems in many P2P systems (see Section 2.6). The certificate is an electronic document that binds a public key to a set of identity information (e.g., name, organization, host name, domain name and node ID) by using a digital signature. To guarantee that the identity in a certificate is the owner of the corresponding public key, the certificate issuer (trusted agent) must verify the identity and the private-key possession before issuing the certificate. Centralized Trusted Third Parties (TTP) and Webs of Trust/PGP [137] are two common ways of issuing certificates. The centralized-TTP method, i.e., a Certification Authority (CA), is more efficient and scalable than the method that uses Webs of Trust. However, the centralized TTP can be a single point of failure, and its existence contradicts the nature of purely decentralized P2P models, which present no centralized TTP. In these P2P application models, each node is equivalent in terms of functionalities (i.e., can simultaneously be both a client and a server) and trustworthiness (i.e., has neither a security hierarchy nor a centralized TTP). Meanwhile, P2P applications are expected to gain reliability, efficiency, and performance from resources sharing among nodes in the same application community. The trust equality among nodes hinders security provision of authentication, integrity, non-repudiation, etc. This situation necessitates a self-organizing and self-healing security system without a centralized TTP for P2P applications.

Thus, we propose a new framework for the **self-organizing and self-healing CA group (SOHCG) in CAN**. Our framework leverages a CAN overlay network with over-

loading zone, an (n, n) threshold signature scheme [114], the homomorphic property of cryptographic functions, and distributed algorithms to construct a CA group. The CAN-based structure facilitates cooperation in the trusted group for certificate issue, and the overloading zone of CAN replicates the key shares of the (n, n) threshold signature scheme to address the lack of fault tolerance in this signature scheme. In our framework, the CA group is a single trusted group and issues certificates for every P2P node. To realize self-organizing and self-healing functionalities, the CA group itself maintains its CAN structure in a dynamic fashion by predefined group management policies, and eliminates malicious nodes from the CA group by a Byzantine Agreement (BA) protocol. To prove an identity and private-key possession, the CA group constructs another CAN overlay network (multicast group) on top of itself, to optimize the communication cost of our protocol. The multicast members, which come from distinct locations on the overlay network, cooperate to prove a network identity and private-key possession with challenge-response messages over disjointed paths. Theses proofs are weighted under a threshold value to resist both internal and external attacks before a certificate is issued.

This framework is composed of a CA Group (a trust group), group management policies, a bootstrapping phase, protocols in the running phase, a certificate revocation protocol, and a key share rendering protocol, all of which will be described in this chapter. The rest of this chapter is organized as follows. Section 4.1 introduces the related work about security in P2P systems according to cryptographic approaches. Section 4.2 describes the model of networks and adversaries, as well as the notations used in the protocol description. Section 4.3 presents the structures used to create a trust group and a multicast group, defines management policies to control the quality and quantity of the trust group, and describes a bootstrapping procedure for initiating a trust group. Section 4.4 describes a suite of protocols to prove node ID and private-key possession, including issuing a certificate. Section 4.5 shows a protocol for key share refreshment in a trust group. Section 4.6 presents a protocol for certificate revocation. Section 4.7 analyzes our framework from two perspectives: attacks and protocols. Finally, Section 4.8 summarizes the result of our framework.

4.1 Related Work on Securing P2P Systems

In the early stages of P2P research, most work focused on making P2P systems feasible in terms of services, reliability and performance. Therefore, both unstructured and structured P2P networks explained in Section 2.3 have security provisions. When P2P networks became widely used on the Internet, P2P users suffered from various kinds of at-

tacks, e.g., viruses, Trojan horses, fake contents, etc. Some researchers have used classical security schemes based on a centralized Trusted Third Party (TTP) to distribute long-term keys for node or content objects. These long-term keys are necessary materials in both symmetric-key and asymmetric-key cryptography, including Certificate Based Cryptography (CBC) and Identity Based Cryptography (IBC), to provide security functions. Hence, a centralized TTP can cope with the lack of security properties in partially centralized and hybrid decentralized P2P systems that have a centralized server or super nodes. However, purely decentralized P2P systems, in which all nodes are equivalent, self-organized and self-maintained, cannot rely on a centralized TTP. As a result, a purely decentralized P2P node lacks a long-term key, which is the basic material for security functionalities. Consequently, researchers have proposed fully decentralized security schemes that are suitable for purely decentralized P2P systems. This section briefly describes some relevant work and points out advantages and disadvantages.

4.1.1 Symmetric-key Cryptography Approach

The preshared key (PSK) is a long-term key commonly used for symmetric-key cryptography because of its simplicity. Its main drawbacks are that every node in the same domain of a P2P system shares the same key, so this key is meant for a domain, not for an individual node. Moreover, if one node is compromised, the whole system is breached. The solution for these drawbacks is to use one distinct PSK for each possible pair of nodes in the system. Hence, the number of keys in n P2P nodes are equal to $N(N-1)$ keys; this large number of keys is burdensome for key distribution. A Key Distribution Center (KDC), such as a Kerberos [121] server, can distribute symmetric keys in order to provide this solution for partially decentralized or hybrid decentralized P2P systems.

In[7], Arora and Shyamasundar proposed PGSP, a P2P authentication protocol using Tamper Proof Hardware (THP) to distribute a unique user ID for each user and a shared Group Access Code (GAC) that is the same for all users participating in the same P2P network. The THP also bundles generate and verify functions for MAC, which is used in key exchange. Users can carry the THP and plug into any nodes joined to a P2P network. Each node will generate a pair of RSA public keys by itself and perform an authentication and a key exchange by using the MAC, which is then generated and verified by the THP. After a public-key exchange, a sender generates a DES key and transmits to a receiver for secure communication.

Although this work can use a pre-shared key (GAC) and ID kept securely in a THP to achieve mutual entity authentication, it needs specific hardware tokens and is based on the

assumption that no one can forge these tokens. Because of the requirement of a THP, this work is limited to use only by federated P2P networks, and the assumption that the THP is unforgeable might not be a realistic assumption. In addition, this solution still needs a centralized administration to initialize the THP before a new node joins the network, and if the GCA shared by all nodes is compromised, the whole network is breached.

4.1.2 Asymmetric-key Cryptography Approach

A public/private key pair is a pair of long-term keys that perform the security function in asymmetric-key cryptography: a private key is concealed, but a public key is revealed, so the authenticity of a public key is the crucial aspect for its security functionality. Two methods are used to maintain the authenticity of public keys: certificate-based and identity-based cryptography. For certificate-based cryptography, the remarkable functions are certification, revocation and renewal of public keys associated with an identity, time and private-key possession. Two traditional approaches are used to achieve these functions: a centralized TTP, i.e., certificate authority (CA), and PGP [137].

Certificate Based Cryptography (CBC) Approach

PGP is an approach that can be deployed instinctively in the purely decentralized architecture of P2P. This approach does not require central servers and uses trusted chains, i.e., trusted relations from node to node for obtaining a required public key. This trusted chain depends on the trustworthiness of the nodes in P2P systems, the human evaluation of key ownership and the difficulty of finding the required public key because a routing map for finding the node holding a required public key does not exist. These problems prevent its large-scale use in P2P systems. In[125] Takeda *et al.* proposed a Hash-based Distributed Authentication Method (HDAM), which deploys PGP: Web of Trust and a Distributed Hash Table (DHT) for mutual authentication between two nodes in a P2P network. This idea is used to provide a decentralized database that depends on trusted relations between peers. The DHT is a routing map for finding the location of the required public key in a distributed database. From the cooperation between the Web of Trust and the DHT, a node can retrieve a required public key by iteratively searching its authentic nodes, i.e., other nodes that it knows their public keys, and so on, until it arrives at the node holding the required public key. In this work, the authors use Chord structure (a hash-ring overlay network), which requires $O(\log_2 N)$ storage per node for maintaining public keys and $O(\log_2 N)$ messages for communication, to retrieve the public key. When node A requests

an authentication from node B, B requires A's public key for this authentication. Therefore, B has to search for A's public key with $O(\log_2 N)$-maximum steps. In each step, B has to authenticate the next node until it arrives at the node holding A's public key. For self-organization, when a new node p joins the network, it has to find a neighbor and provide physical proof to that neighbor, before p obtains the public key of its successor S from the neighbor. Next, p obtains the public keys of its other further successors from S. Finally, S tells its predecessors to update their finger tables. When an existing node needs to leave the network, its must tell its predecessors to update their finger tables. As a result, $O(\log_2 N)$ communication messages are required for a new node to join and leave the network.

This approach works more efficiently than a conventional Web of Trust, whose maximum requirements are $O(N)$ storages for cashing or $O(N)$ communication messages for searching. However, this work still depends on the physical proof of a single node in joining into the network, and if one node in a trusted chain is malicious, this trusted relation will be breached. The authors propose that public keys should be updated periodically with $O(\log_2 N)$-maximum messages to compensate for the lack of malicious node detection and certificate revocation.

In[96] V. Pathak and L. Iftode proposed Byzantine fault tolerant public key authentication that can prove the possession of a private key under an honest majority. This approach does not require a trusted third party, and the authentication is correct if no more than $\lfloor \frac{n-1}{3} \rfloor$ of the n parties are malicious or faulty. This work consists of three main parts: a key authentication protocol, membership control protocol and Byzantine agreement protocol. After a bootstrapping procedure, a trusted group is initiated, and each node maintains its own list of probationary, trusted and untrusted groups. When node A sends an authentication request message to node B, B puts A in the list of B's probationary group and forwards this request message to B's trusted peers. The trusted peers use challenge-response (C-R) messages to prove A's private-key possession, and then return the proofs to B. B can determine A's private-key possession from the validity of C-R pairs. The validations of C-R pairs may not have consensus because A or some trusted peers may be malicious. In this case, B requests C-R pairs from A to prove A's honesty and uses the Byzantine agreement to detect a malicious node in the trusted peers. Moreover, they use a membership control protocol to control the consistency and the size of groups. For example, the validity of key authentication will promote node A from the probationary group to the trusted group. In contrast, malicious behaviors will demote trusted peers from the trusted group to the untrusted group.

The main drawback is that proof of private-key possession creates a trust relation for individual peers' view. Hence, trust relations are not reusable or transitive to other peers

in the P2P system.

In [23] Chen *et al.* proposed an improvement on the Byzantine fault tolerant public-key authentication proposed by V. Pathak and L. Iftode. In this work, they replace the classical challenge-response with sign chaining that optimizes the number of messages in the Byzantine agreement step. The evaluation shows that the number of messages decreases significantly, but the authors provide no security evaluation.

Identity Based Cryptography (IBC) Approach

IBC is another method for maintaining the authenticity of public keys by using an existing known identity as a public key, so the system does not have to maintain certificates. From this characteristic, IBC is called a certificateless scheme. However, IBC requires a centralized TTP, named a Key Generation Center (KGC), for generating private keys, so IBC is not suitable for purely centralized P2P.

In[87], Nguyen proposed a P2P authentication and key exchange protocol that uses the Diffie-Hellman protocol for its session key exchange and IBC for its pairwise shared key distribution. This IBC is based on Weil pairing using two functions:

- Localizing function $L(Q_i, s)$, which a KGC uses to generate a peer's private key $d_i = L(Q_i, s)$ from the KGC's private key s and the peer's public key Q_i (i.e., hashed from its identity ID_i to a point of an additive group on an elliptic curve).
- Mating function $M(d_i, Q_j) = M(d_j, Q_i) = \hat{e}(Q_i, Q_j)^s$, which is used to compute the pairwise shared key $K_{ij} = M(d_i, Q_j) = M(d_j, Q_i)$ between peer i and peer j.

After each peer is preloaded with a private key generated from function$L$ by a KGC, each peer can compute a pairwise shared key by using the function M. This pairwise key is used as a long-term key to authenticate each entity mutually by using a challenge-response message exchange and Message Authentication Code (MAC). The Diffie-Hellman algorithm is used to generate a session key and is combined with MAC to protect against man in the middle (MITM) attacks. In addition, this scheme can extend authentication from a single domain to multiple domains by chaining authentication from one IBC domain to another IBC domain and so on. This chaining authentication is performed by a Trusted Authentication Gateway (TAG), which is a node belonging to two or more IBC domains. This chaining is not a hierarchical system like PKI; therefore, this system can be distributed in any topologies and achieves redundancy load balancing by employing multiple TAGs per IBC domain.

This work can provide efficient and scalable authentication and key exchange in P2P systems; however, KGC is not applicable to purely decentralized P2P, and TAG is equivalent to a centralized TTP (CA) in PKI for each domain; therefore, if a TAG is malicious, it can masquerade as the entity that it authenticates, then deploy MITM attacks.

4.1.3 Decentralized TTP Approach

A TTP is a single entity that holds key material for generating a long-term key, i.e., a CA holds a signature-key for issuing certificates, and a KGC holds a master private key for generating the private keys of peers. This TTP is a single point of failure and key compromise. If the TTP has to serve the entire system, it causes a limitation of scalability in the system. In addition, it is not applicable for purely decentralized and self-organized P2P (a node of the system dynamically joins and leaves) because every node is equivalent and its availability is volatile. Even though some systems can deploy a hierarchical approach for each sub domain, the TTP is a single point of vulnerability in each sub domain. Because of the previous stated facts, it is crucial to devise a decentralized TTP, i.e., CA or KGC, for a purely decentralized P2P system. Threshold secret share or secret share is a common cryptographic system used to achieve this goal.

In[69], Kong *et al.* proposed certificate-related security services for ad-hoc wireless networks that have no infrastructure. This scheme distributes CA functionality to each neighbor node by using the threshold share scheme. A coalition of k neighbors jointly provides a certificate for a requesting node. In the bootstrapping phase, k or more initial nodes are preloaded with share keys SK_i of a CA's signature key SK. In the running phase, a self-initialization protocol handles dynamic node membership, i.e., a new joining node can obtain a CA's secret share from the coalition of k neighbors before becoming a qualified share holder to provide a certificate to a requester without revealing SK_i to the requester. Each shareholder generates and sends a partial certificate signed by its SK_i to the requester. The requester combines all partial certificates to create a complete certificate signed by SK. This work uses RSA cryptography and Lagrange interpolation; therefore, a new certificate X signed by a share key SK_i becomes a partial certificate X^{SK_i}. Combining all partial certificates is the product of partial certificates ($X^{SK_1} \cdot X^{SK_2} \ldots \cdot X^{SK_k} = X^{SK_1+SK_2+\ldots+SK_k} = X^{SK}$) and the k-bounded coalition offsetting algorithm. For self-initialization, a new share key SK_n of a new node k is the summation of partial share keys $SK_i(n)$ ($SK_n = \sum_{j=1}^{k} SK_i(n) = \sum_{j=1}^{k} f(x_i) \cdot l_i(n)$, where $l_i(n)$ is the Lagrange coefficient in the k-coalition at value n.

Even though this work can provide a decentralized CA, in terms of computing and communication load, it take k times more load than traditional CA. The certificate issue

in this work requires physical proof from the neighbors of a new node, and the Certificate revocation is provided by flooding Certificate Revocation Lists over the network. Moreover, periodic updating of all share keys is required to resist comprising of the CA's signature key by k or more collaborative intruders (collusion attack).

In[32], Deng *et al.* proposed a distributed key management and authentication scheme for ad hoc networks that deploys identity-based cryptography and a threshold scheme. This work is based on the assumption that all keys are generated and maintained in a self-organized way, and an IP address or an identity is unique and unchangeable during the lifetime of a network. In this work, the KGC's public key is well know in the whole network, and the KGC's private key is distributed by a (k, n) threshold scheme. Therefore, a coalition of k or more shareholders form a KGC to generate a peer's private key according to that peer's public key Q_{ID}.

In the bootstrapping phase, every initial node selects a secret value x_i, a polynomial $f_i(x)$ of degree $k-1$ such that $f_i(0) = x_i$. Each initial node also generates the KGC's sub-share of private keys s_{ij} for all initial nodes $j = 1, \ldots, n$ and sends them to all other initial nodes $j \neq i$. After each initial node j collects n sub-shares s_{ij} for $i = 1, \ldots, n$, the initial node j can compute KGC's private-key share $s_j = \sum_{i=1}^{n} s_{ij} = \sum_{i=1}^{n} f_i(j)$ and broadcasts a KGC's public-key share $s_j P$, where P is a common parameter of IBC. Consequently, every node can compute KGC's public key by using $\sum_{i=1}^{n} s_i P$. In the running phase, a coalition of k nodes can generate peer's private-keys share $sk_i = s_i Q_{ID}$, where Q_{ID} is a peer's public key according to ID and $i = 1, \ldots, k$. The requester collects all k peer's private-key shares and computes a peer's private key $sk = \sum_{i=1}^{k} s_i Q_{ID}$. For self-organization, when a new node p joins the network, it finds a coalition of k neighbor nodes and requests them new KGC's sub-shares $s_{in} = s_i \cdot l_i(n)$ for p. p computes its KGC's private-key share $s_n = \sum_{i=1}^{k} s_{in}$ by combining all new KGC's sub-shares.

The idea behind this work is similar to Kong's work except for the change in cryptography from CBC to IBC; therefore, the computing load and communication load are k times more than a single KGC's load. The authors also propose a solution to initiate KGC's private-key shares in the bootstrapping phase. Similarly, a collusion attack can compromise a KGC's private key if k or more adversaries collaborate together before the key share updating is finished. However, in private-key generation, an identity verification requires physical proof just as centralized KGC does, and no solution is provided for concealing a private-key share being conducted to a requester.

In[9] Avramidis *et al.* proposed Chord-PKI, a distributed PKI embedded into the Chord overlay network. A Chord-PKI provides a certificate to the Chord nodes themselves without external PKI. The main idea is to partition the Chord network into multiple areas

and empower some nodes in each area with certification functionality, making them into certification nodes. Thereafter, each certification node serves a single area in order to limit the consequences if a certification node is compromised. By employing a (t, n) threshold cryptography, the coalition of the certification nodes in each area can provide the resilience of key compromises and the load-balance of cryptographic functions. The certificate and CRL repository exploit the distributed storage and data management of the Chord network. Thus, storage cost and data manipulation are balanced among the Chord nodes. To obtain a certificate from a Chord-PKI, a Chord node must send a certificate request to one of the certification nodes in the same area. This certification node serves as the combiner of threshold signatures. However, the authors did not provide a mechanism for finding and selecting a certification node and generating a key share for each certification node. A CRL is requested on accusations from a number of certificated nodes, then the CRL is generated by a coalition of certification nodes in the same manner as a certificate.

In this work, the number of certification nodes is static and may be compromised. Although the partitions of Chord address spaces are sized equally, the density of Chord nodes may not be balanced, so the load and security effect of certification and revocation may not be balanced either. There is a trade-off between certificate management and security resilience because with increased security resilience, the number of areas and public-key certificates must increase.

In[73] Lesueur *et al.* proposed a fully distributed certification mechanism based on the trusting of $t\%$ of nodes. A certificate must be signed by the collaboration of $t\%$ of nodes. To achieve this goal, they used the homomorphic property of RSA functions to combine the partial signatures by multiplication. To maintain the ratio $t\%$ of nodes in a collaboration, the number of secret shares must increase according to the increase of the number of nodes in a P2P network. Each share is uniquely identified by a ShareID, which is a binary prefix of a Node ID, i.e., $NodeID = *ShareID$. These ShareIDs are used to form a logical tree structure for searching all $t\%$ combinations of the secret shares that can combine to be a complete signature key. When a new node needs a certificate, it generates a certificate request and sends it to every leaf node on the tree by traversing in an depth-first search. The last leaf node signs the certificate request and sends it back to the previous leaf node, which combines the received partial signature and its own partial signature to create a more complete partial signature, and so on. Finally the new node can combine the received partial signature with its own partial signature for a fully complete signature, i.e., a complete certificate. With this mechanism, if one node in the tree is malicious, the combination of certificates is not successful. To cope with this problem, the authors proposed that each step of the tree traverse should be repeated.

Fully distributed and self-organized characteristics are the strong points of this work.

However, to maintain the security level while a network grows up, the computing load, communication bandwidth and latency time for certification must be increased. As a result, this scheme may not be scalable to an Internet scale.

In this section, we have reviewed certain work, related to three approaches on securing P2P systems or applicable for P2P systems: symmetric key-based, certificate-based and identity-based. Most of these papers work on assumptions that are unsuitable for purely decentralized P2P systems on an Internet scale. The decentralized TTP approaches presented in Subsection 4.1.3 are summarized in Table 4.1.

Table 4.1: The comparison of previous work for CBC in P2P systems

Model	Technique	Limitation	Reference
Hardware	uses Tamper Proof Hardware (THP) to distribute a unique user ID and a domain-shared key	only for private network	[7]
PGP	uses PGP and DHT to create a trusted chain	chain is breached if only one node is malicious	[125]
Individual trust	creates a trust relation by proof of private-key possession under an individual trust group	trust relation cannot be transitive	[96]
Chord PKI	associates each Chord partition with (t, n)-threshold signature	certification node is static	[9]
t% PKI	certificate signatures based on the trust of t% of nodes in the system	not suitable for large scale systems	[73][74]

4.2 System and Security Model

The SOHCG framework presented in this chapter is based on the following models and assumptions.

4.2.1 Structured P2P Network Model

We consider a P2P network as an application-level overlay network over the underlying Internet without IP multicast services. The inter-communication among nodes is asynchronous, which does not guarantee message delivery, reliability, and ordering. The P2P network is composed of dynamic nodes that unpredictably join, leave, and fail. Each node has a unique non-zero network ID mapping tightly to the overlay-network topology, namely a structured P2P overlay network. The multicast and broadcast communication controlled by a CAN-based flooding algorithm [104] can reduce duplicated messages but do not provide Reliable and Totally Ordered (RTO) delivery. For unicast traffic, we assume that a node can be reached by different nodes with distinct paths over overlay networks. We assume that timed services and access to hardware clocks allow time-out and retransmission to mask low level communication failures and provide timed asynchronous communication [29], which is practically implementable for distributed services and provides a notification if an end-point does not exist.

4.2.2 Trusted Group Model

In our trust model, a node is trustworthy if it is trusted by a dynamic trust group, termed a **CA group**, which acts like an internal TTP for a P2P community and forms the security foundation. While none of the nodes trusts others, each node believes that the honest majority of nodes in the trusted group are retained. Since a BA algorithm is used to detect malicious nodes in a CA group, the CA group of size n is expected to have an honest majority if at most $\lfloor \frac{n-1}{3} \rfloor$ malicious nodes exist. Thus, the trusted group can guarantee the consistency of the mapping between a node ID and its corresponding public key in a certificate. Note that we assume that an honest node can protect a private key well and execute the protocols correctly, whereas a dishonest node may behave in any arbitrary fashion.

4.2.3 Bootstrapping Model

Each bootstrap (BT) node has already registered a public key with an external TTP. Thus, it can use cryptographic functions for confidentiality, integrity, and authentication. We assume that all BT nodes can protect their private keys well, execute algorithms correctly, and run cryptographic functions to establish secure and authentic communications among the BT nodes. Moreover, the number of malicious BT nodes is not large enough to reveal

any secret information. For instance, the RSA-key generator, proposed by Boneh and Franklin [16], can prevent collusion attacks from $\lfloor \frac{n-1}{2} \rfloor$ out of n BT nodes to factor the modulus N or to compromise the private key.

4.2.4 Adversary Model

An adversary may either omit to do what it is supposed to do or behave arbitrarily. More specifically, an adversary can eavesdrop on, drop, forge, or man-in-the-middle (MITM) messages. Moreover, we assume that an adversary has limited computational power to break cryptographic functions, i.e., an adversary cannot counterfeit a digital signature and invert an encryption function, but he/she can send any messages at any time. The impact of an adversary also depends on the status of nodes. Thus, we classify adversaries into two categories

External adversary: An adversary who participates in a user overlay network but is not a member of a CA group. This adversary may eavesdrop on, drop, forge or MITM traffic between users and a CA group.

Internal adversary: An adversary who participates in a user overlay network and is a member of CA group. This adversary may drop/MITM traffic among CA nodes, omit to follow the protocol, or launch inconsistent messages in the CA group (see detail in Subsection 4.3.2).

4.2.5 Attack model

Our intention is to devise a CA group for structured P2P networks. Thus, we focus on three main attacks to the CA group:

Node impersonation: An external adversary A' tries to convince a CA group that its public key $PK_{A'}$ belongs to an honest node A. We assume that each P2P node will be reached by different CA nodes through distinct communication paths. Therefore, if adversaries MITM traffic from the CA group on more than a fraction α of paths, then the adversaries can impersonate other nodes.

CA functionality interference: An internal adversary, who has convinced a CA group of his honesty and joined the CA group, may prevent the CA group from issuing certificates to legitimate users by counterfeiting certificate requests, broadcasting invalid challenge causing invalid (c, r) pairs, sending faked partial certificates, or omitting to send partial certificates.

CA group subversion: An internal adversary may intentionally make false accusations to eliminate honest nodes from a CA group, thereby subverting the honest majority of the CA group.

External attacks: Nodes in the underlying network can launch general network attacks, e.g., eavesdropping, Denial of Services (DoS), Sybil, etc.

4.2.6 CA Node Misbehavior Model

Although CA nodes are certified nodes and the messages among CA nodes are signed by their private keys, these certified nodes may become a malicious nodes after they join a CA group and perform bad behaviors in the CA group as follows.

Omission: A CA node may choose not to follow the specified steps in the protocol, e.g., sending a challenge message, reporting a proof result, signing a partial certificate. Omitting to send a challenge message or proof result can be detected by our modified BA algorithm (see details in Subsection 4.4.3). To mitigate omission to sign partial certificates, a coordinator will reestablish a decision group or a requester will resubmit the certificate request (see detail in Subsection 4.4.2).

Message drop: A malicious node drops the messages that use it as an intermediate router to the destination. To mitigate this problem, we use a modified CAN flooding algorithm (see detail in Subsection 4.3.1) that can provide multi-path routing for broadcasting over a CA group and a decision group.

CA interference: A CA node may prevent other CA nodes reaching an agreement by sending forged *CertReq* messages, 2) invalid *Challenge* messages, or 3) forged partial certificates. We need different techniques to cope with these interferences: First, BT nodes sign *CertReq* messages before forwarding the messages to a CA group (see details in Subsection 4.4.1). Second, our *Malicious node Detection Protocol* uses a BA algorithm to detect invalid *Challenge* messages (see details in Subsection 4.4.3). Third, a coordinator

will reestablish a decision group or a requester will resubmit the certificate request (see details in Subsection 4.4.2).

Unlive node: It is a basic operation of the CAN overlay network (CA group) that every node periodically sends refresh messages about its assigned zone to its neighbors and peers. If peers or neighbor nodes do not receive refresh messages from a node in the assigned time, such a node will be considered faulty. Thus, the peers and neighbor nodes will remove such nodes from their peer lists or neighbor lists respectively (see details in Subsection 2.4).

4.2.7 Design Goals

The goal of our framework is to build up a CA group in semi-trusted P2P systems in which P2P nodes do not trust others but believe that the majority of P2P communities is trustworthy. This CA group is delegated by the P2P communities to verify the node ID and public key of P2P nodes before issuing certificates. To this end, the CA group must maintain the following properties:

Self-organizing: A CA group is initialized by BT nodes and then grows to a mature state by itself, thereby requiring neither centralized nor external CA. The membership in the CA group is dynamic and has a uniform distribution over the P2P community.

Self-healing: The honest majority of a CA group is maintained by the CA group itself. All key shares of the CA group are refreshed gradually and continuously, and replicated for key tolerance.

Scalability: The size and construction of a CA group is controlled by predefined parameters for trade-off between security and efficiency.

Resiliency: A CA group has capability to resist or mitigate both attacks from the members of the CA group and the external attacks from nodes in P2P communities.

Efficiency: The computation and communication cost from being a volunteer CA node does not interrupt the functionalities of general P2P applications.

To achieve these goals, we propose a framework for a self-organizing and self-healing certificate authority (CA) in a Content Addressable Network (CAN) that consists of a CA group structure, group management policies, bootstrapping and running phase protocols, a share rendering protocol, and a certificate revocation protocol.

4.2.8 Notation

The notations in Table 4.2 are used for the public-key cryptography, threshold scheme, and message formats in the following protocol descriptions.

Table 4.2: Notation for protocol descriptions

Notation	Description
r_X	A pseudo random number generated by entity X
SK_X	Private(Secret) key of entity X
PK_X	Public key of entity X
SK_X^i	The i^{th} share associated with X's private key
PK_X^i	The i^{th} share associated with X's public key
$U_1, U_2, ..., U_n$	A string generated from a concatenation of strings $U_1 \|\| U_2 \|\| ... \|\| U_n$
$E_{PK_X}(U)$	The cypher text of a string U encrypted with X's public key
$D_{SK_X}(U)$	The plain text of a string U decrypted with X's private key
$Sig_{SK_X}(U)$	A digital signature over a string U generated with X's private key
$[U]_{SK_X}$	A string U attached by digital signature $Sig_{SK_X}(U)$
$CertReq_X$	A certificate request is a binding between X's identifier and public key, signed with its own private key, i.e., $CertReq_X = [X, PK_X]_{SK_X}$
$Cert_X$	A public-key certificate is X's CertReq, including CA's policies, signed with CA's private key, i.e., $Cert_X = [CretReq_X, TimeExp]_{SK_{CA}}$
$Cert_X^i$	A partial certificate is X's certificate request, signed with the i^{th} share of CA
$\rightarrowtail$	Unicast traffic flows over the underlying network
$\rightarrow$	Unicast traffic flows over a user overlay network
$\hookrightarrow$	Unicast traffic flows over a CA overlay network
$\looparrowright$	Unicast traffic flows over a decision overlay network
$\hookrightarrow *$	Broadcast traffic flows over a CA overlay network
$\looparrowright *$	Multicast traffic flows over a decision overlay network

4.3 SOHCG Construction

In this section, we present the structure of a trusted group, termed a **CA group** and the coalition of nodes from the CA group. A CA group is composed of certified nodes, termed **CA nodes**, that are recruited from a P2P network. Hence, some of the CA nodes may be malicious and try to violate the functionality and the honest majority of the CA group, i.e., a CA node may lie or omit to do what it is supposed to do. Consequently, we must define group management policies to control the quantity and quality of the CA group.

4.3.1 A CA Group Structure

The structure of a CA group is a d-dimensional Cartesian space (CAN structure) with overloading zones [103], in which each zone of the CAN overlay network maintains a share corresponding to the CA's private key. Moreover, the CA group recruits certified nodes, (CA nodes), from a user overlay network to form a CAN overlay network. To do so, we must modify the original CAN-based structure and algorithms as follows. Note that nodes sharing the same zone are termed **peers**.

A combination of an (n, n) threshold scheme and a CAN with overloading zones: Duplicating a share with multiple peers and matching a share to a zone can provide fault and key-compromise tolerances. That is, an adversary must disrupt all peers in one zone to disable a CA group or compromise every share (at least one node in each zone) to reveal the CA's private key.

Flooding in a CAN with overloading zones (zone flooding): Two modifications to the multi-path flooding algorithm proposed in [104] are required. First, to provide multi-path routing, we allow CAN nodes to forward a message farther than half-way along the dimension but not to forward a message whose sequence number has already been received. Second, to prevent collusion attacks, all peers of the sender's zone contribute in generating a random CAN ID by using a distributed and synchronized scheme, such as the protocol proposed by Benaloh [12] or Ben-or [13]. According to the distance in the CAN structure, the next zone's peer closest to the random ID is selected as a designated node for each flooding step. Therefore, in each step and so on, a sender cannot intentionally forwards messages to its conspirators. Here, for broadcasting and multicasting, the multi-path flooding is used for non-overloading zones; meanwhile, the zone flooding is used for overloading zones.

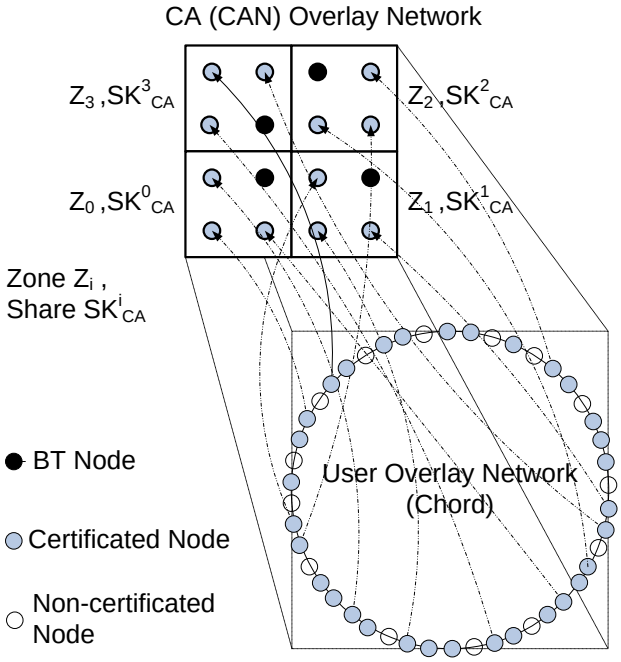

Figure 4.1: Matching CA group to 2-dimensional CAN with overloading

We now involve two categories of P2P networks: one used for constructing a CA group is termed a **CA overlay network**, and one run by users can be any structured P2P overlay network and is termed a **user overlay network**. For the sake of simplicity and without loss of generality, this paper assumes that the CA overlay network is constructed by a 2-dimensional CAN network, and the user overlay network is a Chord network. Figure 4.1 shows the related constructions of both CA and user overlay networks. Note that our framework can extend to support multiple user overlay networks per CA group.

Hence, a CA node participating in both overlay networks must run the protocol stacks of both CA and user overlay networks. Figure 4.2 shows protocol stacks in a user node, a BT node and a CA node, including communication paths between each kind of node.

While the cooperation of CA nodes frequently generates multicast traffics, these traffics can be controlled efficiently by a CAN-based multicast group, named a **decision group**. This decision group is formed by a representative of each zone, named a **decision node**. The decision group is a temporary coalition of CA nodes that proves private-key possession and signs a certificate. Then the decision group disappears after the certificate issue process is done (successful or failed).

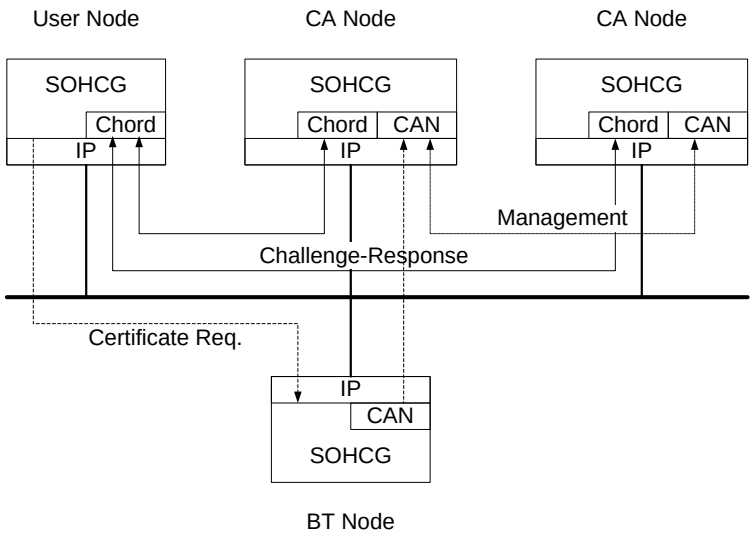

Figure 4.2: Protocol stacks in an user node, a BT node and CA node

4.3.2 Group Management Policies

Since our proof of private-key possession is based on the assumption that paths from different CA nodes to a P2P node are distinct, CA nodes must be selected from a user overlay network uniformly. Moreover, to mitigate collusion and Sybil attacks [38], CA nodes' membership must be dynamic in order to limit the time, i.e., *age* rounds of certificate issuing, that a malicious CA node can subvert the functionality and honest majority of a CA group. To achieve the above two properties, we define the following two policies.

Peer Recruitment

When the number of peers in one zone is less than p_{min}, all peers in that zone cooperate to generate a random ID with the same distributed scheme employed in zone flooding (See Subsection 4.3.1). The peer closest to the random ID is selected as a coordinator. Thus, all peers are equally likely to be selected, instead of a fixed coordinator per zone. Based on a look-up service in the user overlay network, the coordinator selects the user node associated with the random ID. The coordinator then verifies whether the selected node is a certified node, neither a current CA node nor a revoked node, before inviting this node to be a new CA node. If the verification fails, another random ID will be generated until peers can find a satisfactory node. In the joining step, the coordinator hands over the

parameters and share to the new CA node via a secure and authentic channel established by the public-key scheme. These parameters and shares are also verified by compared to the ones from other neighbor nodes. Unlike to the original CAN, in which a new node requests to join CAN, our joining step is initiated from the cooperation of peers to select a new node. As a result, every peer has control of and participation in peer recruitment.

Peer Retirement

In a CAN with overloading zones, every CAN node (or CA node) maintains a peer list and a neighbor list (i.e., a list of peers in its adjacent zones). Usually an entry of these two lists is composed of a node ID and its IP address. We add a *Cert* field (i.e., the certificate of that node) and a *CertCount* field (i.e., a counter of the number of certificates issued by that node) to the peer list. Additionally, we add only a *Cert* field to the neighbor list. When a CA node P is selected to be a member of a decision group, every peer of P queries P's entry in its peer list and decreases the value of P's *CertCount* field. If the *CertCount* field is greater than zero, P is selected. Otherwise, P's peers consider P as an **expired node**, remove P from their peer lists, and tell P's neighbors to remove P from their neighbor lists as well. Hence, P is retired from the CA group. Moreover, in a CAN overlay network, every node must periodically send refresh messages to its neighbors and peers. If P's peers or neighbors do not receive the refresh messages within a limited time, they consider P an **unlive node** and remove P from their peer lists or neighbor lists, respectively.

With certificate provision and group management policies, P2P nodes can be classified according to their functionalities (i.e., user nodes, CA nodes, or decision nodes), behavior (i.e., malicious or non-malicious), and certification status (i.e., certified or non-certified), as shown in Figure 4.3. Nodes in each group may overlap those in other groups and migrate from one group to others according to their certification procedures (i.e., certificate issuing or revocation) and the group management policies (i.e., retirement or recruitment).

Although our peer retirement and peer recruitment policies cause nodes to continuously join and leave the CA group (or the CAN overlay network) at a high churning rate, a CAN overlay network can still operate well because only $2d$ zones are affected by the joining or leaving of one node (where d is the dimension of a CAN overlay network). Before a CA group is formed and maintained by a coalition of certified nodes, the CA group must be initialized by BT nodes in the bootstrapping phase, as described in the next subsection.

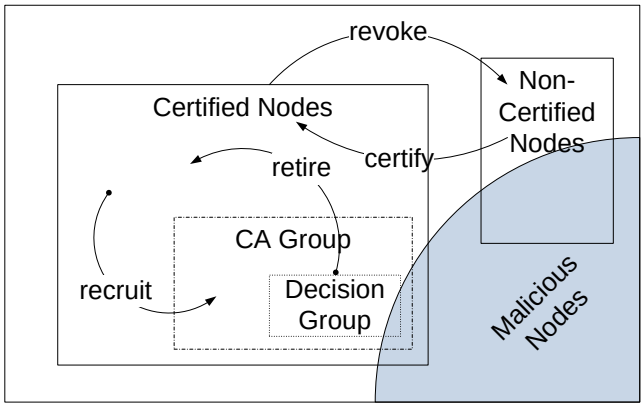

Figure 4.3: Group relation and management

4.3.3 Bootstrapping Phase

At the beginning of this phase, the authority creates BT nodes to form a CA group and to function as shareholders in a threshold signature scheme. Although a large number of peers and zones in a CA group can increase fault tolerance and attack resistance, increasing a CA group's size trades efficiency for security. In other words, if the number of peers in one zone increases, not only does fault tolerance increase but also the number of states (neighbors and peers lists). Meanwhile, if the number of zones (shares) increases, not only does the key-compromised tolerance increase but also the computing and communication cost. To craft our scheme for a tradeoff between efficiency and security, we define the following parameters to control CA group size.

p_{min}**:** the minimum number of peers before recruitment.

p_{max}**:** the maximum number of peers before splitting a zone, where $p_{max} \geq 2p_{min} + 2$.

z_{min}**:** the minimum number of zones at the initial state.

z_{max}**:** the maximum number of zones at the mature state, where $z_{max} \geq 2z_{min}$.

age**:** the maximum number of certificates each node can issue before retirement.

To initialize an (n, n) threshold signature scheme without a dealer and knowledge about the corresponding private key and to establish a CA group according to the predefined parameters, the bootstrapping phase consists of the following two steps.

Zone and Share Initialization

The authority constructs a 2-dimensional CAN with z_{min} zones and at least z_{max} BT nodes. Hence, an initial CA group, which starts with z_{min} zones (at least two BT nodes per zone) and a (z_{min}, z_{min}) threshold scheme, has a suitable and practical size for a start-up network. Then z_{min} BT nodes from different zones use Boneh and Franklin's algorithm [16] to select a public key PK_{CA} and to generate a modulus N and shares SK^i for each zone. Both the modulus N and the public key PK_{CA} are published on public servers or embedded in software distributions, as happens in many operating systems and web browsers. Additionally, the IP addresses of the BT nodes are published under the CA domain name or in the extension field of the CA certificate [56]. The BT nodes then start issuing certificates for P2P users and managing the CA group.

CA Group Initialization

Meanwhile, the BT nodes start recruiting certified nodes to the CA group and hand over the shares and parameters through secure and authentic channels established by a public-key scheme. When a zone consists of p_{max} peers, the BT nodes in that zone delete their shares and fade out from the threshold signature scheme forever. The CA group now arrives at the initial state, which can grows up by itself until arriving at the mature state. However, the BT nodes are still permanent members of the CA group and function as gateways for submitting certificate requests. Therefore, BT nodes maintain CA policies and current peer-lists for certificate requests. Note that BT nodes now are not members of user overlay networks, so they do not suffer from the attacks from user overlay networks.

Now the CA zone itself can grow up without BT nodes. Each zone continues recruiting until the number of peers amounts to the upper bound p_{max}. Then the zone is split into two halves, each of which contains half of its peers and BT nodes if applicable. To limit the number of zones to z_{max}, the coordinator P_{Z_0} of a split zone broadcasts a *Zone Count* message to the CA group to get zone information from every zone before splitting, as shown in Table 4.3. The coordinator then checks the completeness of the zone combinations in the Cartesian space before counting the number of zones.

If the current number of zones is less than z_{max}, the recruiting and splitting are repeated until the CA group consists of z_{max} zones and p_{max} peers in every zone. For example, if a zone Z_0 has to be split into two new zones Z_{00} and Z_{01}, all peers in zone Z_0 cooperate to select a nonce r in the manner used in zone flooding (See Subsection 4.3.1). Next, zone Z_0 is split into two new zones Z_{00}, Z_{01} with the new shares $SK_{CA}^{Z_{00}} \equiv r \pmod N$ and $SK_{CA}^{Z_{01}} \equiv SK_{CA}^{Z_0} - r \pmod N$, respectively. Finally, zones Z_{00} and Z_{01} render their

Table 4.3: Zone count protocol

Traffic Flow	Message Format / Action (where $i = 1, ..., n$)
$P_{Z_0} \hookrightarrow *$:	$[P_{Z0}, *, Zone\ Count]_{SK_{P_{Z0}}}$
$\{P_{Z_i}\} \hookrightarrow P_{Z_0}$:	$[P_{Z_i}, P_{Z_0}, Zone\ Information, ZoneInf_i]_{SK_{P_{Z_i}}}$
P_{Z_0}:	Combines zone areas and counts the number of zones

new shares with their neighbor zones. The details of the *Share Rendering Protocol* are presented in Subsection 4.5.

Eventually, the CA group grows to a mature state with the number of peers and zones bounded at p_{max} and z_{max} respectively. This mature state is the proper size for achieving a suitable efficiency/security trade-off. In other words, if the number of key shares (zones) in a CA group increases, the key compromised tolerance increases; meanwhile the computing and traffic load of key combination processes increases. When the number of peers in one zone increases, not only does the fault tolerance increase but also the number of states (neighbors and peers) maintained by each node increases.

The following sections explain how the CA group can achieve two main functionalities of a traditional CA, i.e., registration and certificate issuing. We assume that every node in both a CA group and user overlay network knows the CA certificate and the BT nodes' certificates. However, the members of the CA group know only the key shares associated with their zones.

4.4 SOHCG Running Phase Protocols

4.4.1 Key-Registration Protocol

When a new node joins a user overlay network, it must request its certificate from a CA group to upgrade from an uncertified node to a certified one. To this end, a CA group first prepares its delegation, called a **decision group**, when receiving a certificate request. The decision group performs the later protocols and multicast efficiently and employs a challenge-response protocol to prove the network ID and the private-key possession. If the verification succeed, the decision group will generate a certificate and send it to the new node. This protocol consists of the following four steps.

Step 1. Certificate Request

When node A needs a new certificate, A first generates a public/private key pair by itself and constructs a certificate request $CertReq_A = [A, PK_A]_{SK_A}$ under the PKCS 10 [88] format, then submits the $CertReq_A$ to one of BT nodes. The BT node does not issue the certificate by itself but forms a *Forward Request* FR message by attaching CA policies (e.g., expiry time, etc.) to the $CertReq_A$ and signing with the BT node's private key, before randomly forwarding the FR message to one of the CA nodes P_0. The message flows and formats in this step are as follows.

Traffic Flow	Message Format / Action
A:	Generates $CertReq_A = [A, PK_A]_{SK_A}$
$A \rightarrowtail BT$:	$[A, BT, Cert\ Request, CertReq_A]_{SK_A}$
$BT \hookrightarrow P_0$:	$FR = [BT, P_0, Cert\ Request, CertReq_A, ExpTime]_{SK_{BT}}$

Afterwards, the certificate request is submitted to the CA group for verification. The decision group will be established to perform three main functions of a traditional CA: 1) prove the network ID in the certificate request, 2) verify the possession of the private key associated with the proposed public key, 3) sign the certificate in order to guarantee the correspondence between the network ID and the public key, i.e., binding the network ID and the public key in the certificate. These main functions will be performed in the later steps.

Step 2. Decision Group Establishment

The first selected node P_0 (assume that P_0 is in zone 0) combines its node ID and certificate with the *FR* message to form a *Group Request* message and then broadcasts the *Group Request* message to a CA group with zone flooding algorithm. Hence, all nodes selected through the zone flooding algorithm are the decision nodes. To form a decision group with P_0, each decision node responds to P_0 with a *Group Join* message consisting of its zone information and certificate, and stores the *FR* message for generating a partial certificate. After forming a decision group, which is a CAN-based multicast group called a **mini-CAN**, P_0 must check the completeness of the Cartesian space by combining all zones' information from every *Group Join* message. Therefore, the number of nodes (or zones) in a decision group is equal to the number of zones in a CA group. In other words, the decision group has all the shares for reconstructing a complete certificate, as shown in Figure 4.4. Then P_0 learns the IDs and certificates of all decision nodes from the *Group Join* messages. To

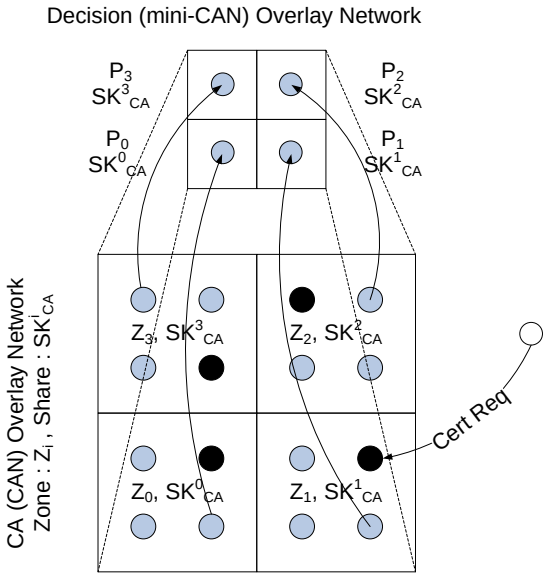

Figure 4.4: Forming a decision group over a CA group

confirm the completeness of the decision group, P_0 multicasts all the information to the decision group with a *Group Confirmation* message. Thus, each decision node knows the others, forms a mini-CAN with P_0 and is ready to prove A's private-key possession. The below diagram shows the message flows and formats in this step.

Traffic Flow	Message Format / Action (where $i = 1, ..., n$)
$P_0 \hookrightarrow *$:	$[P_0, *, Group\ Request, FR, (P_0, Cert_{P_0})]_{SK_{P_0}}$
P_i:	Records FR and forms a mini-CAN with P_0
$P_i \hookrightarrow P_0$:	$[P_i, P_0, Group\ Join, P_i, ZoneInf_i, Cert_{P_i}]_{SK_{P_i}}$
P_0:	Checks the completeness of zone area
$P_0 \not\hookrightarrow *$:	$[P_0, *, Group\ Confirmation, (P_0, Cert_{P_0}), \ldots, (P_n, Cert_{P_n})]_{SK_{P_0}}$
P_i:	Form a mini-CAN with P_0

Step 3. Challenge-Response

Decision nodes, which are placed in distinct locations on a user overlay network, use challenge-response messages to prove A's network ID and private-key possession. Each decision node P_i generates a nonce r_{P_i} and a challenge $C_{P_i} = E_{PK_A}(r_{P_i})$, which is the

nonce encrypted with A's public key, and then sends C_{P_i} to A. A decrypts the challenge using its private key and sends the response $R_{P_i} = D_{SK_A}(C_{P_i})$ to P_i. A also stores these challenge-response $C_{P_i,A}, R_{A,P_i}$ message pairs in a vector $\overrightarrow{V_A}$ for the *Malicious Node Detection Protocol*, which is introduced in Subsection 4.4.3. The message flows and formats in this step are shown in the diagram below:

Traffic Flow	Message Format / Action (where $i = 0, ..., n$)
P_i:	Generates a nonce r_{P_i} and a challenge $C_{P_i} = E_{PK_A}(r_{P_i})$
$P_i \rightarrow A$:	$C_{P_i,A} = [P_i, A, Challenge, C_{P_i}, Cert_{P_i}]_{SK_{P_i}}$
A:	Generates a response $R_{P_i} = D_{SK_A}(C_{P_i})$
$A \rightarrow P_i$:	$R_{A,P_i} = [A, P_i, Response, R_{P_i}]_{SK_A}$
A:	$V_A[i] = (C_{P_i,A}, R_{A,P_i})$

In general, these challenge-response messages are sent over a P2P overlay network. Since the network ID of a structured P2P network is tightly tied to the P2P node's IP address by a hash function, we may send the challenge-response messages over the underlying IP network to prove network ID, while avoiding eclipse attacks in structured P2P networks.

Step 4. (C, R) **Agreement**

The proof of private-key possession is based on an agreement on the validity of (C, R) pairs from every decision node. This agreement can be reached in three levels: **consensus**, **threshold**, and **none**. The values of the validity and the agreement are defined below.

- *The Validity of a* (C, R) *Pair:* Denoted by $val(C, R)$, if $C_{P_i} = E_{PK_A}(R_{P_i})$, then $val(C, R) = ture$, otherwise $val(C, R) = false$.
- *The Agreement on a Decision Vector* $\overrightarrow{V}$*:* The consensus function of $\overrightarrow{V}$, denoted by $consensus(\overrightarrow{V})$, and the threshold function of $\overrightarrow{V}$, denoted by $threshold(\overrightarrow{V})$, are defined as follows.
  - If all $V[i]$ have the same value v, then $consensus(\overrightarrow{V}) = v$, otherwise $\perp$.
  - If at least $n-(\lfloor\frac{n-1}{3}\rfloor+\alpha n)$ of $V[i]$ have the same value v, then $threshold(\overrightarrow{V}) = v$, otherwise $\perp$, where n is a number of nodes in the decision group and $\alpha < 1/3$ is a fraction of paths is attacked by MITM.

We denote a decision vector $\overrightarrow{V} = (v_1, v_2, ..., v_n)$, where $v_i = val(C_i, R_i) \in \{true, false\}$. The three levels of agreement on $\overrightarrow{V}$ are defined as follows.

Definition 3 *We define the following functions.*

$$\begin{aligned} \textit{Consensus:} \quad & consensus(\overrightarrow{V}) = v,\ threshold(\overrightarrow{V}) = v \\ \textit{Threshold:} \quad & consensus(\overrightarrow{V}) = \perp,\ threshold(\overrightarrow{V}) = v \\ \textit{None:} \quad & consensus(\overrightarrow{V}) = \perp,\ threshold(\overrightarrow{V}) = \perp \end{aligned}$$

Each decision node P_i must multicast its $(C_{P_i,A}, R_{A,P_i})$ message pair to the others by a *(C,R) Distribute* message. Each also saves all $(C_{P_i,A}, R_{A,P_i})$ message pairs as a vector $\overrightarrow{V_{P_i}}$. If there is a true threshold on $\overrightarrow{V_{P_i}}$, then P_i runs *Certification Issue Protocol*, which will be presented in Subsection 4.4.2. If the threshold on $\overrightarrow{V_{P_i}}$ is false, then P_i sends an *Authen Fault* message to P_0. However, if there is no consensus on (C, R) pairs ($consensus(\overrightarrow{V}) = \perp$), a decision node runs the *Malicious Node Detection Protocol* (described in Subsection 4.4.3). The diagram below shows all the message flows and formats in this step.

Traffic Flow	Message Format / Action (where $i = 0, ..., n$)
$P_i \nrightarrow *$:	$[P_i, *, (C, R)\ Distribute, (C_{P_i,A}, R_{A,P_i})]_{SK_{P_i}}$
P_i:	$V_{P_i}[j] = (C_{P_j,A}, R_{A,P_j})$
	For j from 1 to n
	$\quad \overrightarrow{V}[j] = val(\overrightarrow{V_{P_i}}[j])$
P_i:	If $threshold(\overrightarrow{V}) = true$
	$\quad$ Runs *Certificate Issue Protocol*
	ElseIf $threshold(\overrightarrow{V}) = false$
	$\quad P_i \nrightarrow P_0$: $[P_i, P_0, Authen\ Fault]_{SK_{P_i}}$
	If $consensus(\overrightarrow{V}) = \perp$
	$\quad$ Runs *Malicious Node Detection Protocol*

Generally, if either A or a decision node P_i lied or has been MITMed by an adversary more than a threshold value, a decision group will not issue A's certificate. If an agreement on a decision vector does not have true consensus, either A or some decision nodes P_is are malicious. Thus, our protocol has to run the *Malicious Node Detection Protocol* in order to investigate who is a liar. Note that our protocol considers that a node lied or that its messages have been MITMed by MITM attacks as the same misbehavior.

4.4.2 Certificate Issue Protocol

Based on the threshold signature and the homomorphic property of a public-key cryptographic function, each decision node computes a partial certificate in a distributed fashion and sends it to a coordinator who combines all partial certificates and reconstructs a complete certificate. In the following, we use the RSA scheme as an example to explain the approach. Let public key PK be (e, N) and private (secret) key SK be d. Note that for a message m, if $d \equiv d_0 + d_1 + ... + d_k \pmod{\phi(N)}$, then we obtain $m^d \equiv m^{\sum_{i=0}^{k} d_i} \pmod{N} \equiv \prod_{i=0}^{k} m^{d_i} \pmod{N}$. The *Certification Issue Protocol* consists of the following two steps, of which the flows and formats are shown in Table 4.4. Figure 4.5 pictorially shown this protocol.

Step 1. Partial Certificate Generation

Once a decision node P_i has a true agreement on (C, R), i.e., consensus or threshold level; P_i accepts that the proposed node ID and public key are valid. Hence, P_i signs the $CertReq_A||ExpTime$ with its share SK_{CA}^i and sends the partial certificate $Cert_A^i$ to the coordinator.

Step 2. Complete Certificate Construction

This step is run only when a decision node is a coordinator P_0. To construct A's complete certificate, a coordinator P_0 must gets all partial certificates with no *Authen Fault* message from decision nodes. Then the coordinator verifies the $Cert_A$ with the CA's public key. If the verification is successful, P_0 sends the $Cert_A$ to A. Otherwise, some $Cert_A^i$ might be corrupted if some decision nodes are malicious or have compromised shares. Hence, P_0 repeats *Decision Group Establishment Protocol* (see Subsection 4.4.1) to select a new decision group until the verification is successful or the number of runs exceeds a predefined limit. When the limit is exceeded, P_0 informs A by sending a *Cert Fault* message. Although P_0 is malicious and sends A a faulty certificate, A can detect that fault by using CA's public key to verify the certificate. In this case, A resubmits a certificate request to change the coordinator P_0 who may be malicious.

Since a CA group selected from a P2P community may consist of malicious nodes, we need a detection protocol to detect malicious nodes in the CA group.

Table 4.4: Certificate issue protocol

Step 1. Partial Certificate Generation

Traffic Flow	Message Format / Action (where $i = 0, ..., n$)
P_i:	Generates $Cert_A^i = [CertReq_A, ExpTime]_{SK_{CA}^i}$
	$P_i \nrightarrow P_0$: $[P_i, P_0, Partial\ Cert, Cert_A^i]_{SK_{P_i}}$

Step 2. Complete Certificate Construction (only for a coordinator node)

Traffic Flow	Message Format / Action
P_0:	If *no Authen Fault*
	Reconstructs $Cert_A = \prod_{i=0}^{n} Cert_A^i \pmod N$
	Verifies $Cert_A$ with PK_{CA}
	If verification is successful
	$P_0 \rightarrow A$: $[P_0, A, Complete\ Cert, Cert_A]_{SK_{P_0}}$
	Else-If limit is not exceeded
	Runs *Decision Group Establishment Protocol*
	Else
	$P_0 \rightarrow A$: $[P_0, A, Cert\ Fault]_{SK_{P_0}}$
	Else
	$P_0 \rightarrow A$: $[P_0, A, Authen\ Fault]_{SK_{P_0}}$

4.4.3 Malicious Node Detection Protocol

However, our *Key Registration Protocol* does not consider who (either A or P_i) causes invalid (C, R) pairs, and the honest decision nodes have already accepted the proposed node ID and public key from A if the number of valid (C, R) pairs is more than the threshold value. Our decision group still faces a message synchronization problem if P_i sends a invalid challenge messages in the decision process. These invalid messages cannot be detected by using signature schemes because the invalid messages are generated by the private-key owners. Although the invalid messages cannot subvert the *Key Registration Protocol*, our framework should have an algorithm to detect and eliminate malicious nodes from a trust group in order to retain the honest majority of the trust group (CA group).

To maintain that honest majority in a CA group, the *Malicious Node Detection Protocol* must investigate whether some decision nodes lie, send invalid challenge messages, or drop

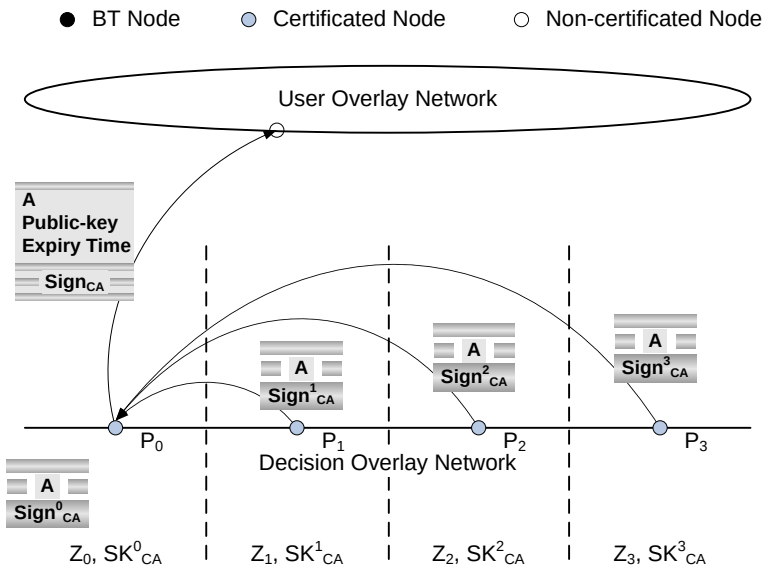

Figure 4.5: Decision nodes cooperate to issue a certificate

messages in order to eliminate such malicious nodes from the CA group. In addition, this detection protocol should be executed only when some decision nodes show suspicious behavior because running a BA algorithm is costly. Consequently, we divide this protocol into the following two steps (see Table 4.5) to optimize computation and communication cost.

Step 1. Proof Request

To avoid running a BA algorithm frequently, our protocol considers A as a witness, so each decision node P_i requests the vector $\overrightarrow{V_A}$ from A by sending a *Proof Request* message. Then P_i compares the $\overrightarrow{V_A}$ with its own vector $\overrightarrow{V_{P_i}}$. P_i investigates the consistency of challenges in $V_A[j]$ and $V_{P_i}[j]$: If there are inconsistent challenges from the decision node P_j, then P_j is a liar in P_i's view because no one can counterfeit the signed challenge from P_j. However, other decision nodes cannot rely on P_i's accusation because they have no knowledge about what P_j said to P_i. Otherwise P_i could make a faulty accusation causing an honest node to be eliminated. Hence, P_i must multicast a *Byzantine Request* message to activate a BA algorithm, which creates useful information so that every honest decision node can reach the same agreement on an accusation of the bad behavior (under the assumption that the upper bound of the number of malicious nodes $f \leq \lfloor \frac{n-1}{3} \rfloor$, where n is the number

of decision nodes). In other words the inconsistency of challenge messages, the invalidity of (C, R) pairs results from P_i's invalid challenge messages. That is, P_i is a liar, and our protocol will request the BA algorithm.

Step 2. Byzantine Agreement

When a decision node receives the *Byzantine Request* message, it multicasts its own vector $\overrightarrow{V_{P_i}}$ with a *Byzantine Agree* message. After receiving all *Byzantine Agree* messages, each decision node runs the BA algorithm (see Subsection 3.1.2) to detect the decision node M providing invalid messages. The adjacent decision nodes of M cooperate to eliminate the malicious node M and to select a new CA node to replace it. Then the decision group repeats the *Key Registration Protocol* from step 3: Challenge-Response.

Table 4.5: Malicious node detection protocol

Step 1. Proof Request

Traffic Flow	Message Format / Action (where $i = 1, ..., n$)
$P_i \rightarrow A$:	$[P_i, A, Proof\ Request]_{SK_{P_i}}$
$A \rightarrow P_i$:	$[A, P_i, Proof\ Response, \overrightarrow{V_A}]_{SK_A}$
P_i:	For j from 0 to n
	If $C_{P_j,A}$ in $V_A[j]$ and $V_{P_i}[j]$ is not consistent
	$P_i \nrightarrow *$: $[P_i, *, Byzantine\ Request]_{SK_{P_i}}$

Step 2. Byzantine Agreement

Traffic Flow	Message Format / Action (where $i = 1, ..., n$)
$P_i \nrightarrow *$:	$[P_i, *, Byzantine\ Agree, \overrightarrow{V_{P_i}}]_{SK_{P_i}}$
P_i:	Run BA algorithm on Challenge messages
$M's\ peers$:	Eliminate M from the CA group
	Reselect a new decision node to replace M
P_i:	Go to Step 3 in Key Registration Protocol

Since we cannot distinguish between an omission node and a crash node. We can consider both kinds of these faulty nodes are unlive nodes that must be rejected from a CA group but that its certificate is not revoked. To detect an unlive node, each node will

assign a specific value $\emptyset$ for challenge(s) that cannot be received before a defined timeout. The BA algorithm then uses a majority function to detect any consistently unlive or lying nodes.

Definition 4 *The majority function of a set of n values is denoted by* $major(v_1, ..., v_n)$*: if there are more than* $\lceil \frac{n+1}{2} \rceil$ *of* $v_i = v$*, then* $major(v_1, ..., v_n) = v$*, otherwise* $major(v_1, ..., v_n) = \perp$*.*

Therefore, if a majority value of challenge vales belongs to which decision node is $\perp$, then that node is a liar, or if the majority value is $\emptyset$, then that node is unlive. An example of the decision arrays of a decision group with the number of nodes $n = 4$ and the number of faulty nodes $f = 1$ when node 3 lies (Example 1) and node 3 omits (Example 2) are shown in Table 4.6 and 4.7, respectively.

Table 4.6: Example 1. The decision matrix when node 3 lies

Round		P_1 received	P_2 received	P_3 received	P_4 received
1^{st}	P_i sent	$[1, 2, w, 4]$	$[1, 2, x, 4]$	$[1, 2, 3, 4]$	$[1, 2, z, 4]$
2^{nd}	P_1 sent	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$	$[1, 2, w, 4]$
	P_2 sent	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$	$[1, 2, x, 4]$
	P_3 sent	$[1, 2, a, 4]$	$[1, 2, b, 4]$	$[1, 2, c, 4]$	$[1, 2, d, 4]$
	P_4 sent	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$	$[1, 2, z, 4]$
$majority(\cdot)$		$[1, 2, \perp, 4]$	$[1, 2, \perp, 4]$		$[1, 2, \perp, 4]$

Table 4.7: Example 2. The decision matrix when node 3 omits

Round		P_1 received	P_2 received	P_3 received	P_4 received
1^{st}	P_i sent	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, 3, 4]$	$[1, 2, \emptyset, 4]$
2^{nd}	P_1 sent	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$
	P_2 sent	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$
	P_3 sent	$[\emptyset, \emptyset, \emptyset, \emptyset]$	$[\emptyset, \emptyset, \emptyset, \emptyset]$	$[1, 2, 3, 4]$	$[\emptyset, \emptyset, \emptyset, \emptyset]$
	P_4 sent	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$
$majority(\cdot)$		$[1, 2, \emptyset, 4]$	$[1, 2, \emptyset, 4]$		$[1, 2, \emptyset, 4]$

In addition to the previous protocols used to achieve the main functionalities of CA, as shown in Figure 4.6, our framework needs another protocol to maintain CA's key shares

in a proactive fashion because CA's public key cannot be changed frequently, and the membership of a CA group is dynamic. In the next subsection, we describe the protocol used to refresh the key shares periodically and to replace the previous shares of eliminated nodes.

4.5 Share Rendering Protocol

Although the *peer retirement* (see Subsection 4.3.2) and the *Malicious Node Detection protocols* (see Subsection 4.4.3) provide the replacements for compromised or malicious CA-nodes, doing so leaks the shares of eliminated nodes. Therefore, our scheme uses the additive inverse property of an (n, n) threshold signature scheme and a shuffling scheme to implement a *Share Rendering Protocol*, which gradually and randomly renders shares according to the retirement, unlive and misbehavior of CA nodes. This protocol is activated by eliminated CA nodes' peers, who cooperate with their neighbor nodes in each dimension, e.g., x- and y-axis for 2-dimensional CAN. Without loss of generality, we explain this protocol in only one dimension and assume that a CA node M in zone 1 is eliminated while holding a share SK^1_{CA}. Therefore, the CA group must replace the share SK^1_{CA} by rendering it with its neighbor's shares (i.e., SK^0_{CA}, SK^2_{CA}). The rendering process is accomplished in the following four steps and graphically shown in Figure 4.7.

Step 1. Coordinator Selection: All peers in zone 1 generate a random number b and select a coordinator C_1, which is the node closest to the random number a, based on the local information in their peer lists. Respectively, zone 0 and 2 generate random numbers a and c, and select coordinators C_0 and C_2. Starting from the middle zone Z_1, the coordinator C_1 learn the node IDs of coordinators C_0, C_2 from C_1's neighbors, then introduces itself and C_0, C_2 to each other. Now each coordinator learns other coordinators and their certificates by exchanging *Coordinator Confirmation* messages $[C_i, C_j, Coordinator\ Confirmation, Cert_{C_i}]_{SK_{C_i}}$.

Step 2. Nonce Exchange: According to the shuffling scheme, each coordinator C_0, C_1, C_2 sends its nonce $n = a, b, c$ to the pairwise coordinator C_1, C_2, C_0, respectively, in such a way that no coordinator knows all nonces. For confidentiality, such nonce is encrypted by the receiver's public key before attaching to a *Nonce Exchange* message $[C_i, C_j, Nonce\ Exchang, E_{PK_{C_j}}(n)]_{SK_{C_i}}$ and being sent to the pairwise coordinator.

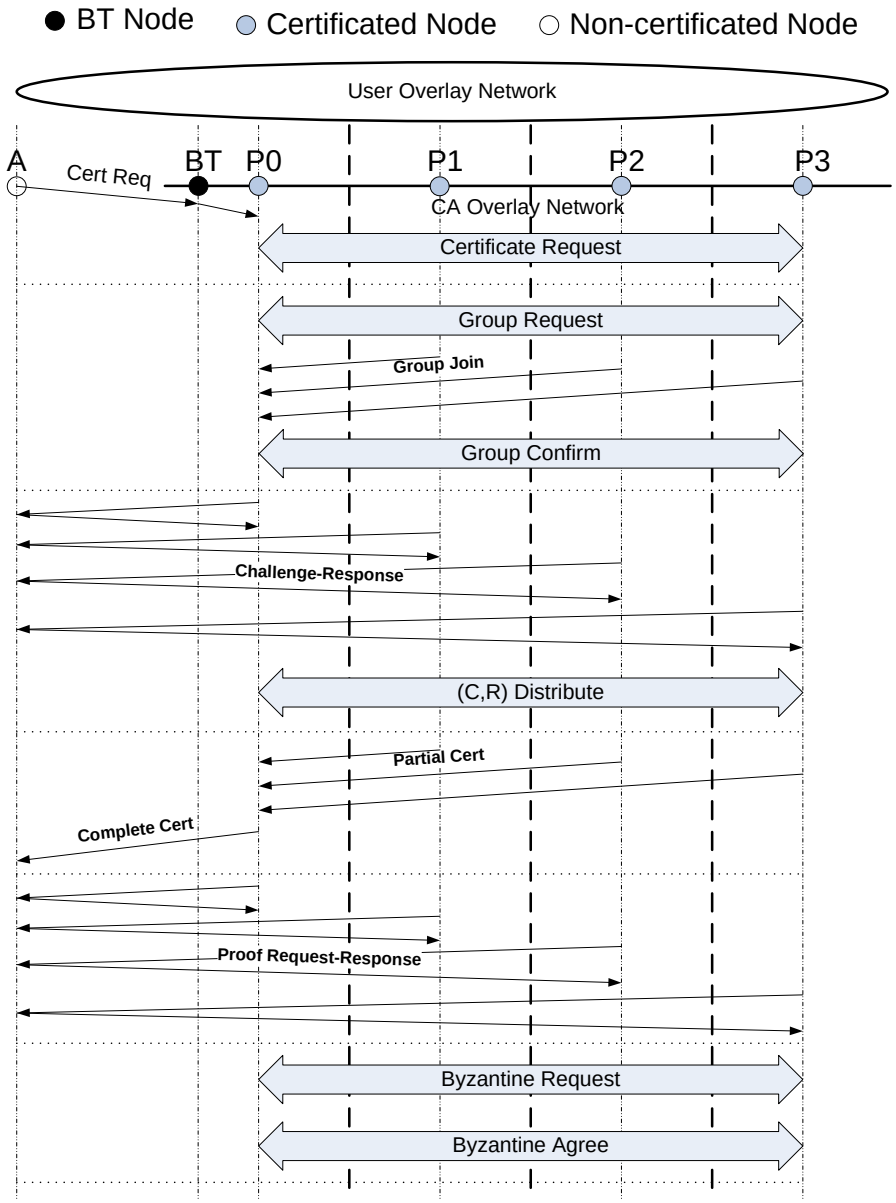

Figure 4.6: Protocols in running phase: node A asks for its certificate and is authenticated by a decision group

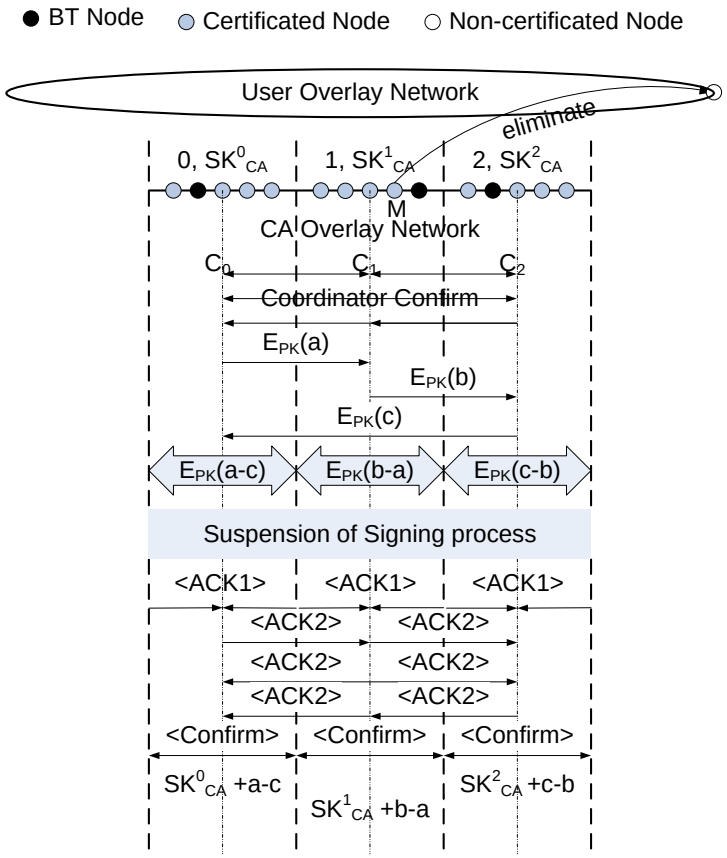

Figure 4.7: Rendering the share: after a malicious node P is rejected

Step 3. Nonce Distribution: Each coordinator C_i computes the difference $diff_i$ between its nonce and the received nonce and sends the $diff_i$ encrypted by the receiver's public key to all its peers $Peer_i$ with *Nonce Distribution* messages $[C_i, Peer_i, Nonce\ Distribution, E_{PK_{Peer_i}}(diff_i)]_{SK_{C_i}}$. To prevent the *Share Rendering Protocol* causing share inconsistency, every peer suspends new certificate requests until all holding requests are finished and then acknowledges its coordinator with *ACK1*.

Step 4. Nonce Confirmation: When the coordinator receives *ACK1* from all its peers, it acknowledges itself and the other two coordinators with *ACK2*. Each coordinator will broadcast a *Nonce Confirmation* message $[C_i, Peer_i, Nonce\ Confirmation]_{SK_{C_i}}$ to all its peers after receiving *ACK2* from all three coordinators. Then every peer renders its share (i.e., $\widehat{SK}^0_{CA} = SK^0_{CA} + a - c, \widehat{SK}^1_{CA} = SK^1_{CA} + b - a, \widehat{SK}^2_{CA} = SK^2_{CA} + c - b$) and continues the suspended processes.

In our scheme, a CA group gradually and randomly renders its shares in each particular part of the CA group (i.e., that zone and its neighbor zones) so that the CA group can continue working and rendering simultaneously. Moreover, adversaries cannot recover a CA private key without holding all shares before any adjacent remainder-shares are rendered.

4.6 Certificate Revocation Protocol

For a certificate-based scheme, a process to revoke a certificate is as necessary as one to issue a certificate. A Certificate Revocation List (CRL) is one method used to revoke a certificate in the X.509 standard. In our framework, we consider using a CRL to revoke the certificate of a malicious node in order to reject and prohibit attacks from the malicious node. Moreover, we use a benefit of the structured P2P network to store the CRL on the user overlay network by using certificates as a key to identify the storage location, e.g., for Chord, the identity of a CRL is $SHA(Cert)$.

Refer to the *Malicious Node Detection Protocol*, if a node M is accused of being a malicious node during the byzantine agreement step (see Section 4.4.3), the malicious node M should be eliminated from a CA group. To this end, the honest decision nodes will cooperate in issuing and signing a revocation request *RevReq* by using the same method they use to sign a certificate request. This *RevReq* is also used as evidence if M's peers and neighbors are asked to eliminate M from the CA group (the CAN structure).

For the sake of simplicity, we assume that node M in zone Z_1 has been identified as a malicious node. Then a coalition of honest decision nodes generates and distributes a *RevReq* as shown in Figure 4.8. This protocol is performed in the following five steps.

Step 1. CA Node Selection: After the malicious node M is detected, the decision node in M's neighbor zone will cooperate to select another CA node Q from M's zone by using the same distributed scheme as is used in the zone flooding (see Subsection 4.3.2). Node Q joins a decision group by replacing M and can cooperate in the threshold signature scheme because it occupies the same share as M.

Step 2. Revocation Request: The coordinator P_0 (or its neighbor if P_0 itself is a malicious node) creates a revocation request $RevReq = [Cert_M, Trev]_{SK_{P_0}}$, where $Cert_M$ is the certificate of the malicious node M and $Trev$ is the time of revocation. Then, P_0 broadcasts the $RevReq$ over the decision group.

Step 3. Partial CRL: Each decision node generates a partial CRL $CRL_M^i = [RevReq_M]_{SK_{CA}^i}$ by signing a $RevReq$ with CA's key share SK_{CA}^i and sends the partial CRL to P_0.

Step 4. Complete CRL: P_0 creates a complete CRL $CRL_M = [RevReq]_{SK_{CA}}$ by producing its partial CRL CRL_M^i with the other partial CRLs, i.e., $CRL_M = \prod_{i=1}^{k} CRL_M^i$.

Step 5. CRL Distribution: P_0 stores a CRL on the user overlay network (assume that it is a Chord overlay network) by using a basic function of the structured overlay network, e.g., the $put(key, value)$ function and the $value = get(key)$ function of the Chord overlay network. For instance, a $put(SHA(Cert_M), CRL_M)$ is used to store M's (Cert, CRL) pair and a $get(SHA(Cert_M))$ is used to retrieve the CRL of M, as shown in Figure 4.8. Thus, the malicious node M cannot join the CA group until it can obtain a new certificate. Moreover, P_0 sends the complete CRL to all M's peers and neighbors to ask them to delete M from their peer lists and neighbor lists. Thus, M is eliminated from the CA group, and M's certificate is revoked by CRL.

Moreover, since nodes in a user overlay network might be malicious, we cope with this problem by storing replicated CRLs in many locations on the user overlay network. This replication can be achieved by using embedded feature of the overlay network or using multiple hash functions to map $Cert_M$ to many locations.

4.7 Security Analysis

This section analyzes the security of our framework in two approaches: attacks and protocols. First, three main attacks to the proposed framework, i.e., node impersonation, CA

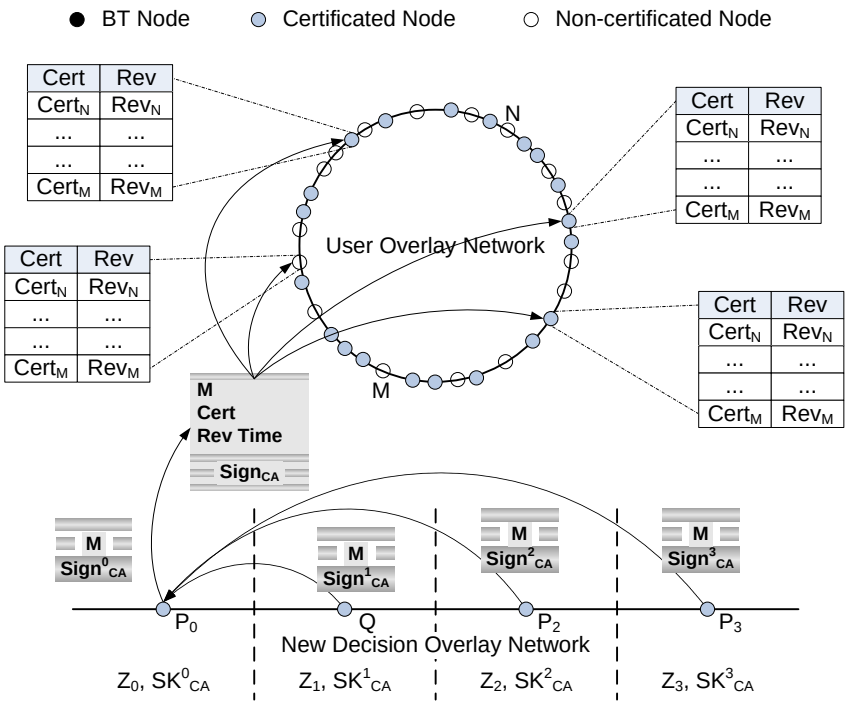

Figure 4.8: A coalition of the decision nodes cooperates in CRL distribution

functionality interference, and CA group subversion, and then other possible attacks are analyzed. Second, two main protocols, i.e., key registration and malicious node detection, are analyzed under the boundary of a parameter and a BA algorithm.

4.7.1 Attack Analysis

Node Impersonation Attacks: In this attack, an external adversary, malicious node A' or a coalition of malicious nodes $\{A'_i\}_{i=1,\ldots,n}$ try to impersonate a legitimate node A in two cases:

1. A malicious node A' may propose a certificate request $[A, PK_{A'}]_{SK_{A'}}$ to a CA group. Because the majority of challenge messages from a CA group will reach A instead of A', most of the proofs of private-key possession will fail because the public/private key pairs of A and A' are different. Therefore, adversary A' fails in impersonation.

2. A coalition of malicious nodes $\{A'_i\}_{i=1,\ldots,n}$ may launch MITM attacks to compromise the *Key Registration Protocol*. $\{A'_i\}_{i=1,\ldots,n}$ collude to launch MITM attacks in order to MITM and counterfeit A's certificate request and challenge-response messages. These messages are forged to convince the CA group that one of the adversaries is A and possesses A's private key. However, the collusion of $\{A'_i\}_{i=1,\ldots,n}$ cannot succeed in impersonating A if they cannot MITM more than αn challenge-response messages.

CA Functionality Interference: We consider the case of internal adversaries (malicious decision-nodes) that try to interfere three CA group functionalities: key registration, key authentication and certificate issue as follows.

1. Key registration: When a new node sends a certificate request to a trusted gateway (a BT node) known by all the decision nodes, the BT node signs both the user's certificate request and CA policies before forwarding them to all the decision nodes. Hence, our framework can guarantee that every decision node receives the same proposed public key and policies because no one can modify or fabricate the registration information.

2. Key authentication: A malicious decision-node may distribute invalid challenges to corrupt the challenge-response scheme and cause a legitimate user to fail in its key-possession proof. Our protocol leverages the threshold value $(n - \lfloor \frac{n-1}{3} \rfloor - \alpha n)$, which includes the number $(\lfloor \frac{n-1}{3} \rfloor)$ of the internal adversary under the upper bound of the

BA algorithm. Therefore, our scheme can resist both internal and external adversaries under the upper bound.

3. Certificate issue: A malicious decision-node may omit a partial signature or create a wrong one to interrupt certificate issuing. Although our framework cannot verify a partial signature to detect the malicious decision node, re-establishing a new decision group or re-submitting a certificate request can mitigate this attack.

CA Group Subversion: Without global view, a malicious decision-node may intentionally make a false accusation causing a CA group to eliminate an honest decision-node. The *Malicious Detection Protocol* deliberately uses signed message and the Byzantine algorithm to discover which decision nodes cause challenge-response invalidity. This detection is based on the knowledge built from messages between members in the decision group, instead of individual knowledge; therefore, the adversary cannot convince a decision group to believe false accusations against honest decision-nodes.

DoS Attacks: A malicious user-node A may send a decision group invalid responses to activate the BA algorithm. To thwart the attacks, each decision node P_i compares the challenges in its own vector $\overrightarrow{V_{P_i}}$ with the ones in A's vector $\overrightarrow{V_A}$ to confirm whether decision nodes lie before activating the BA algorithm. Consequently, the adversary cannot exploit the BA algorithm to disable the services of the CA group but may launch DoS attacks in other scenarios.

Sybil Attacks: Although our framework cannot prohibit an adversary from acquiring multiple IDs, it can limit the effect of Sybil attacks on the CA group. On the basis of a *CertCount* field in a peer list, a *Retirement* policy and an *age* parameter, the framework can limit the lifetime of a Sybil ID in the CA group. In addition, decision nodes are selected randomly from the CA nodes, which are selected uniformly from a P2P community in a distributed manner. Hence, the Sybil nodes cannot use their multiple IDs to subverting the CA group's honest majority.

Reveal the CA private key: Since eliminated or retired nodes still occupy their shares after leaving from a CA group, the CA group disables such shares by using the *share rendering protocol*. Hence, adversaries cannot gain more the knowledge of the CA private key from the shares they occupied.

4.7.2 Protocol Analysis

Registration Protocol

We give the security analysis of our protocols together with some examples to illustrate the attacks and countermeasure.

1. Let A be MITMed by adversaries $\{A'_1, \ldots, A'_{\alpha n}\}$, where α denotes a fraction of MITMed path, and n denotes the number of decision nodes or zones.

2. Hence, a decision group $\{P_1, \ldots, P_i\}$ may get a certificate request $CertReq_A = [ID_A, PK_A]_{SK_A}$ from A, or $CertReq'_A = [ID_A, PK_{A'}]_{SK_{A'}}$ from A'. In the former case, the response from A will be valid, but that from A' will be invalid. The later case, the response from A will be invalid, but that from A' will be valid. However, under the upper bound α, a decision group will accept the proposed certificate request from A, but reject the one from A'.

3. The decision group proves the proposed node ID and private-key possession of the requester in the name of A by using the proposed public key $PK = PK_A$ (or $PK_{A'}$) to generate challenges $\{C_1, \ldots, C_4 | C_i = E_{PK}(r_i)\}$ (where r_i is a random number generated by P_i), which are sent to the requester node with destination node ID = A, from each decision node in the district location on a P2P network.

4. Therefore, A will receive $(1-\alpha)n$ challenges from decision nodes, while $\{A'_1 \ldots A'_{\alpha n}\}$, which are intermediate nodes on the P2P-network path between A and the decision nodes, can MITM α challenges from the decision nodes. Then A replies with responses $R_i = D_{SK_A}(C_i)$, while $\{A'_1 \ldots A'_{\alpha n}\}$ reply with responses $R_i = D_{SK_{A'}}(C_i)$. That is, the responses must be generated by using the private key that A or A' has.

   Example 3. An example for illustrating step 4: Assume that $n = 4$, $\alpha = 1/4$, P_2 is a dishonest decision node under Byzantine bound $\lfloor \frac{4-1}{3} \rfloor = 1$, and P_3 is an MITMed decision node, i.e., a decision node who has been MITMed by A' under $\alpha \cdot n = (1/4) \cdot 4 = 1$.

CA Node	Challenge	A' Response	A Response
P_1	$C_1 \rightarrow$		$\leftarrow R_1$
P_2	$C'_2 \rightarrow$		$\leftarrow R_2$
P_3	$C_3 \rightarrow$	$\leftarrow R'_3$	
P_4	$C_4 \rightarrow$		$\leftarrow R_4$

Note that R'_3 is generated with $PK_{A'}$ not PK_A

5. Each decision node must share its Challenge-Response (C, R) pair with the other decision nodes by broadcasting its (C, R) pair over the decision group. Therefore, every decision node will obtain all (C, R) pairs and maintain its own (C, R) vector $\overrightarrow{(C, R)}$, in order to verify the node ID and the private-key possession in the succeeding steps.

   Example 4. An example for illustrating step 5: Assume that P_2 is dishonest by sending challenge C_2 to P_3 and invalid challenge C'_2 to P_1, P_4; each decision node will get the $\overrightarrow{(C, R)}$ as follows:

	P_1 received	P_2 received	P_3 received	P_4 received
P_1 sent	$[C_1, R_1]_{11}$	$[C_1, R_1]_{12}$	$[C_1, R_1]_{13}$	$[C_1, R_1]_{14}$
P_2 sent	$[C'_2, R_2]_{21}$	$[C_2, R_2]_{22}$	$[C_2, R_2]_{23}$	$[C'_2, R_2]_{24}$
P_3 sent	$[C_3, R'_3]_{31}$	$[C_3, R'_3]_{32}$	$[C_3, R'_3]_{33}$	$[C_3, R'_3]_{34}$
P_4 sent	$[C_4, R_4]_{41}$	$[C_4, R_4]_{42}$	$[C_4, R_4]_{43}$	$[C_4, R_4]_{44}$

   Note that P_2 can broadcast invalid challenges in the decision group because flooding over a multicast CAN is not a totally ordered and reliable broadcast.

6. Each decision node computes the validity of a (C, R) pair, denoted by $val(C, R)$, if $C_{P_i} = E_{PK}(R_{P_i})$, then $val(C, R) = true$, otherwise $val(C, R) = false$. Then each decision node will get decision vector $\overrightarrow{V} = (v_1, \ldots, v_n)$, where $v_i = val(C, R)$. For simplicity, we will analyze only the case in which $CertReq$ comes from A ($PK = PK_A$) in the later steps.

   Example 5. An example for illustrating step 6: Assume that the certificate request $CertReq_A = [ID_A, PK_A]_{SK_A}$ is proposed by A, namely $PK = PK_A$. The decision vector of each decision node is computed as follows:

$\overrightarrow{V}_{P_1}$	$(\overrightarrow{V}_{P_2})$	$\overrightarrow{V}_{P_3}$	$\overrightarrow{V}_{P_4}$
true	–	true	true
false	–	true	false
false	–	false	false
true	–	true	true

Note that the decision vector $\overrightarrow{V}_{P_2}$ cannot be determined because we assume that P_2 is a malicious node.

7. The decision node will locally prove the node ID and the possession of the private-key associated with the proposed public key PK by using his own decision vector and threshold function. Moreover, the decision node also uses a consensus function to detect misbehavior, in order to decide whether a malicious node detection protocol should be run. Recall that the threshold function and the consensus function defined in Definition 3.

   Example 6. An example for illustrating step 7: The threshold value $= 4 - (1 + 1) = 2$, so the proof of the proposed node ID and the private-key possession are output as the following results:

	$\overrightarrow{V}_{P_1}$	$(\overrightarrow{V}_{P_2})$	$\overrightarrow{V}_{P_3}$	$\overrightarrow{V}_{P_4}$
	true	–	true	true
	false	–	true	false
	false	–	false	false
	true	–	true	true
threshold(V)	true	–	true	true

   In conclusion, except for the malicious node P_2, all honest decision-nodes P_1, P_3, P_4 accept the proposed node ID A and public key PK.

8. Each decision node can detect which response is improperly signed, and ignores the $val(C, R)$ of such responses from his decision vector $\overrightarrow{V}$. In other words, the decision node can detect which responses come from A or A'.

   Example 7. An example for illustrating step 8: From the vector $\overrightarrow{(C, R)}$ in step 5 and vector $\overrightarrow{V}$ in step 6, if the signed response message R'_3 is invalid, the pair $[C_3, R'_3]$ can be ignored. Hence, the vector $\overrightarrow{(C, R)}$ and $\overrightarrow{V}$ are updated and shown in the following table.

P_1 received $\rightarrow \overrightarrow{V}_{P_1}$	P_2 received $\rightarrow \overrightarrow{V}_{P_2}$	P_3 received $\rightarrow \overrightarrow{V}_{P_3}$	P_4 received $\rightarrow \overrightarrow{V}_{P_4}$
$[C_1, R_1]_{11} \rightarrow$ true	$[-,-]_{12} \rightarrow$ $-$	$[C_1, R_1]_{13} \rightarrow$ true	$[C_1, R_1]_{14} \rightarrow$ true
$[C_2', R_2]_{21} \rightarrow$ false	$[-,-]_{22} \rightarrow$ $-$	$[C_2, R_2]_{23} \rightarrow$ true	$[C_2', R_2]_{24} \rightarrow$ false
$[C_3, R_3']_{31} \rightarrow$ $-$	$[-,-]_{32} \rightarrow$ $-$	$[C_3, R_3']_{33} \rightarrow$ $-$	$[C_3, R_3']_{34} \rightarrow$ $-$
$[C_4, R_4]_{41} \rightarrow$ true	$[-,-]_{42} \rightarrow$ $-$	$[C_4, R_4]_{43} \rightarrow$ true	$[C_4, R_4]_{44} \rightarrow$ true
consensus(V) $\perp$	$-$	true	$\perp$

However, no consensus about the value of a decision vector means that either A or some decision nodes are liars.

Malicious Node Detection Protocol

Although the decision nodes P_1, P_3, P_4 can use the proposed public key to verify the signed message from A, they still suffer from a message synchronization problem in the decision group. For instant, the column vector $\overrightarrow{(C, R)}$ of the table in Example 7 shows that honest decision nodes P_1, P_4 still have an invalid $[C_2', R_2]$ pair from P_2. The question is who causes this invalidity. The answer is whether A or P_2 lies will cause the same invalidity, and all challenges and responses are properly signed by the sources of messages. Thus, the signature cannot detect who is a liar. To cope with this problem, the decision node must investigate whether P_2 lies, by the following steps:

1. Each decision node P_i can prove whether P_i lies by asking A to directly send vector $\overrightarrow{(C, R)}_A$ to the decision node. If the challenges in $\overrightarrow{(C, R)}_A$ is equal to the ones' in $\overrightarrow{(C, R)}_{P_i}$, then P_i is not a liar (the invalidity is caused by the responses from A). Hence, it is not necessary to run the detection protocol. Otherwise P_i lies and necessitates running the *Malicious Node Detection Protocol*.

   Example 8. An example for illustrating step 1: With the comparison between $\overrightarrow{(C, R)}$ of A and P_i, P_2 and P_4 realize that P_2 is a liar because P_2 sent C_2 to A and P_3, but sent C_2' to P_1, P_4. The following tables show the comparisons between column vectors of P_i with the one from A.

P_1 received	P_2 received
$[C_1, R_1]_{11} : [C_1, R_1]_A$	$[C_1, R_1]_{12} : [C_1, R_1]_A$
$[C'_2, R_2]_{21} : [C_2, R_2]_A$	$[C_2, R_2]_{22} : [C_2, R_2]_A$
$[C_3, R'_3]_{31} : [C_3, R'_3]_{A'}$	$[C_3, R'_3]_{32} : [C_3, R'_3]_{A'}$
$[C_4, R_4]_{41} : [C_4, R_4]_A$	$[C_4, R_4]_{42} : [C_4, R_4]_A$

P_3 received	P_4 received
$[C_1, R_1]_{13} : [C_1, R_1]_A$	$[C_1, R_1]_{14} : [C_1, R_1]_A$
$[C_2, R_2]_{23} : [C_2, R_2]_A$	$[C'_2, R_2]_{24} : [C_2, R_2]_A$
$[C_3, R'_3]_{33} : [C_3, R'_3]_{A'}$	$[C_3, R'_3]_{34} : [C_3, R'_3]_{A'}$
$[C_4, R_4]_{43} : [C_4, R_4]_A$	$[C_4, R_4]_{44} : [C_4, R_4]_A$

2. Since P_2 is a malicious node, it may send any arbitrary challenge value to outvote other honest decision nodes. Hence, the honest decision nodes must convince each other and agree that P_2 sent the invalid challenge C'_2. To do so, decision nodes P_1, P_4 broadcast Byzantine requests over the decision group to force every decision node to broadcasts its own vector $\overrightarrow{(C, R)}$. Note that detection protocol is interested in only challenge C_i, not response R_i. Recall that the majority function and the consensus function defined in Definition 4.

   Example 9. An example for illustrating step 2 From the previous table, P_2 causes the decision node P_3 to locally decides that P_2 did not lie, but P_1, P_4 locally decide that P_2 lies. Therefore, P_1 and P_4 broadcast Byzantine requests to the decision group to force every decision node to broadcast its own (C, R) vector. Under the majority value $\lceil \frac{4+1}{2} \rceil = 3$, all honest decision nodes P_1, P_3, P_4 can use a majority function to detect the malicious node P_2 as follows.

	P_1 received			
P_1 sent	$[C_1]_{111}$	$[C'_2]_{211}$	$[C_3]_{311}$	$[C_4]_{411}$
P_2 sent	$[W_1]_{121}$	$[W_2]_{221}$	$[W_3]_{321}$	$[W_4]_{421}$
P_3 sent	$[C_1]_{131}$	$[C_2]_{231}$	$[C_3]_{331}$	$[C_4]_{431}$
P_4 sent	$[C_1]_{141}$	$[C'_2]_{241}$	$[C_3]_{341}$	$[C_4]_{441}$
$majority()$	C_1	$\perp$	C_3	C_4

	P_3 received			
P_1 sent	$[C_1]_{113}$	$[C'_2]_{213}$	$[C_3]_{313}$	$[C_4]_{413}$
P_2 sent	$[Y_1]_{123}$	$[Y_2]_{223}$	$[Y_3]_{323}$	$[Y_4]_{423}$
P_3 sent	$[C_1]_{133}$	$[C_2]_{233}$	$[C_3]_{333}$	$[C_4]_{433}$
P_4 sent	$[C_1]_{143}$	$[C'_2]_{243}$	$[C_3]_{343}$	$[C_4]_{443}$
$majority()$	C_1	$\perp$	C_3	C_4

	P_4 received			
P_1 sent	$[C_1]_{114}$	$[C'_2]_{214}$	$[C_3]_{314}$	$[C_4]_{414}$
P_2 sent	$[Z_1]_{124}$	$[Z_2]_{224}$	$[Z_3]_{324}$	$[Z_4]_{424}$
P_3 sent	$[C_1]_{134}$	$[C_2]_{234}$	$[C_3]_{334}$	$[C_4]_{434}$
P_4 sent	$[C_1]_{144}$	$[C'_2]_{244}$	$[C_3]_{344}$	$[C_4]_{444}$
$majority()$	C_1	$\perp$	C_3	C_4

From the above results, decision node P_1, P_3, P_4 can locally decide that P_2 is a liar because they cannot get a majority value for P_2's challenge.

4.8 Summary

In this chapter, we proposed SOHCG, which is a fully self-organizing and self-healing system based on a CAN overlay network. In our scheme, a CA group grows up under a security/efficiency trade-off, maintains its membership in a dynamic fashion and employs a challenge-response scheme to provide an automatic key registration and a certificate issue. Meanwhile, a CAN with overloading zones compensates for the lack of fault tolerance in an (n, n) threshold signature scheme, and a CAN-based multicast is used to optimize the computation and communication cost of a BA algorithm. In addition, a BA algorithm with sign messages is leveraged to maintain an honest majority with the avoidance of DoS attacks in a CA group. Finally, all CA's shares are refreshed in a gradually random fashion. The security analysis shows that our scheme can prevent impersonation, collusion and MITM

attacks as well as mitigate Dos and Sybil attacks. The time slot for an adversary to reveal the CA private-key is equal to the period of a refreshment, according to the frequency of certificate issues. Below we give s summary for the SOHCG consisting of all protocols and steps in Table 4.8.

Table 4.8: The protocols and steps in SHOCG

SOHCG	
Protocols	Steps
Key registration	1. Certificate request 2. Decision group establishment 3. Challenge-response 4. (C, R) agreement
Certificate issue	1. Partial certificate generation 2. Complete certificate construction
Malicious node detection	1. Proof request 2. Byzantine agreement
Share rendering	1. Coordinator selection 2. Nonce exchange 3. Nonce distribution 4. Nonce confirmation
Certificate revocation	1. CA node detection 2. Revocation request 3. Partial CRL 4. Complete CRL 5. CRL distribution

Chapter 5

OAuth and ABE based Authorization (AAuth)

In computer security, access control is an essential component used to approve the access privileges of consumers. Traditionally, a centralized access control system consists of only two roles: 1) clients who act as consumers on behalf of resource owners, 2) a centralized server which authenticates consumers and authorizes access to resources according to consumers' capabilities or resources' Access Control Lists (ACLs). With the development of Single Sign-On (SSO) systems like Kerberos [68], an authenticator and an authorizer cloud can be separated from a centralized server, while the owner and consumer act in the same client.

In Kerboros, an authenticator will issue a token to a client as an identity proof if the authentication succeeds. Then the client can reuse the token to request access tokens from an authorizer to achieve SSO, until the token expires. Finally, the access token is used to request a protected resource from a resource server that enforces access policies.

Once web applications became ubiquitous on the Internet, sharing resources between HTTP-service providers became an important requirement for both users and providers. For example, Facebook applications use a subscriber's Google address book to look for new friends, or photo labs print a customer's pictures from her Flickr. Although standards exist for authentication and authorization, many applications in practice rely on the assumption that users trust their service providers to keep and process their data. The assumption is rational if we can assume that all parties are in the same trusted domain or owners never expose their sensitive data outside their on-premise storage or data center. With the emergence of cloud computing, however, this assumption may not be true. For example, IT

infrastructures are outsourced to public clouds whose Cloud Service Providers (CSPs) may be dishonest. In this hostile environment, users must trade control of data for flexibility, scalability, and reduced expenses from clouds. Finally, users may decide to lock in a single CSP since they believe it can mitigate concerns about data security. This belief impedes the open-cloud concept, and no one can guarantee security because of the following problems: 1) an authorizer may arbitrarily grant access tokens to its conspirators. 2) resource servers hosting sensitive information may reveal this information. 3) a resource server may refuse to obey predefined capabilities or ACLs. 4) on large scale systems like the Internet, assuming that all participants are in a single trusted domain is infeasible realistic situations.

To dispense with vendor lock-in or the need to hope that CSP is trustworthy, we propose a novel authorization scheme, **ABE-based Authorization (AAuth)**, based on the OAuth authorization standard and Ciphetext-Policy Attribute Based Encryption (CP-ABE). This new scheme has the following features.

1. AAuth employs a user-centric approach by using a web browser preloaded with CA certificates as an HTTP trust-platform for owners. Using our modified CP-ABE scheme adapted from Bethencourt [14], owners bundle an ACL, namely access structures, into protected data by using CP-ABE encryption.

2. Our scheme also replaces the traditional tokens with ABE-based tokens (termed ABE-tokens), which are CP-ABE private keys. Therefore, the ACLs will be enforced end-to-end when consumers try to decrypt the encrypted data with the private keys in the ABE-tokens.

3. To achieve our user-centric approach, we allow owners to limit the lifetime and scope of ABE-tokens by adding additional owner-controlled attributes, called confined attributes. Moreover, we modify CP-ABE key generation so that an authority, an authorizer, and an owner can contribute to the key generation.

4. Owners delegate their shares associated with a time slot to an authorizer who acts as a time server. This delegation allows the authorizer to synchronize the time slot attributes in encrypted files with light-weight and lazy re-encryption, thus gaining efficiency from cloud servers and preventing owners from staying on-line to limit token lifetime.

5. Our design also realizes that the requirement for public-key certificates is inapplicable for Internet end-users, and a single trust-domain is inapplicable and non-scalable in large-scale systems.

With the modification and combination of OAuth, CP-ABE, ElGamal-like masking, proxy re-encryption, and lazy re-encryption, our scheme can achieve user-centric and end-to-end cryptographic functions that support the following functionalities.

1. Access grants are performed by the cooperation among owners, an authorizer, and an authority(s).
2. Access policies are bundled into resources by owners, then enforced at the destination.
3. Data is encrypted when resting with a resource server, then decrypted at the destination.
4. In contrast to previous works that propose completely new designs, our scheme integrates existing standards and available Internet infrastructures for flexibility and compatibility.
5. Moreover, our scheme is designed in a distributed fashion for scalability but can merge related roles into the same entity for simplicity.

The rest of this chapter is organized of follows: Section 5.1 reviews previous work related to cloud storage and access control. Section 5.2 discusses our models and assumptions. Section 5.3 describes definitions and notations for our explaination. Section 5.4 presents our construction, procedures and protocols. Section 5.5 analyses our scheme in term of security from internal and external adversaries. Section 5.6 evaluates the performance of our scheme by using a simulation and compares our scheme to existing standards. Finally, Section 5.7 concludes the paper.

5.1 Related Work on Securing Cloud Computing

The taxonomy of cloud computing services goes by the acronym 'SPI', which stands for Software-as-a-Service, Platform-as-a-Service, and Infrastructure-as-a-service. Recently, many new cloud services have emerged; for a general introduction, see Chapters 2, 3. The most primitive and significant one is Storage-as-a Service. Many researcher are exploring this research area, starting from cryptographic storage to access control to cloud storage. In 2003, Kallahalla *et al.* proposed Plutus, a cryptographic file system. This secure file system exploits cryptographic and key management in a decentralized manner in which all operations are performed by clients, and the server incurs very little cryptographic

overhead. This behavior contradicts cloud-computing behavior in which servers have no limited resources but clients may be restricted devices. Thus, clients may not have enough power for the high complexity of key management in fine-grained access control.

In 2009, Bowers *et al.* [17] proposed the framework for a Proof of Retrievability (POR) system that focuses on archival or backup files in cloud storage. Later, they also proposed another POR work [18] that was implemented by a distributed cryptographic scheme and launched on the multi-server of a distributed file system. Wang *et al.* [138] proposed cryptographic-based access control for owner-write-users-read applications, which encrypts every data block of cloud storage and adopts a key derivation method to reduce the number of keys. Yun *et al.* [134] proposed a cryptographic network file system based on MAC tree construction and universal-hash-based state-full MAC that can guarantee the data confidentiality and integrity of files.

In 2010, Yu *et al.* [133] proposed fine-grained and scalable access control in cloud computing that exploits KP-ABE to reduce complexity in key management and key distribution. They also use proxy re-encryption to off-load cryptographic operations to cloud servers, and lazy re-encryption to reduce cryptographic cost on servers.

In 2011, Zarandioon *et al.* [135] proposed K2C, a scalable ABE-based access hierarchy that can couple access control with the folder structure of file systems. By combining KP-ABC and a key-updating scheme, K2C users can use new keys to decrypt data encrypted with old keys in order to reduce re-encrypting cost on servers when access hierarchies update keys. In addition, the combination of KP-ABE and a signature scheme allows users to prove that others own keys that satisfy the policy in order to provide signing and verification in K2C.

5.2 System and Adversary Models

Currently, access control for on-line transactions in clouds is performed in the form of a token issued by an authorization server and enforced by a resource server with predefined capabilities and/or ACLs. Therefore, all participants in the application domain must trust that the authorizer issues a token only to legitimate consumers and that resource servers honestly grant access according to the access policies of such tokens. This section explains the model and assumptions of our scheme based on adversary models, and the main objectives of our design.

5.2.1 System Model

There are five main parties in the system as pictorially shown in Figure 5.1 and described as follows:

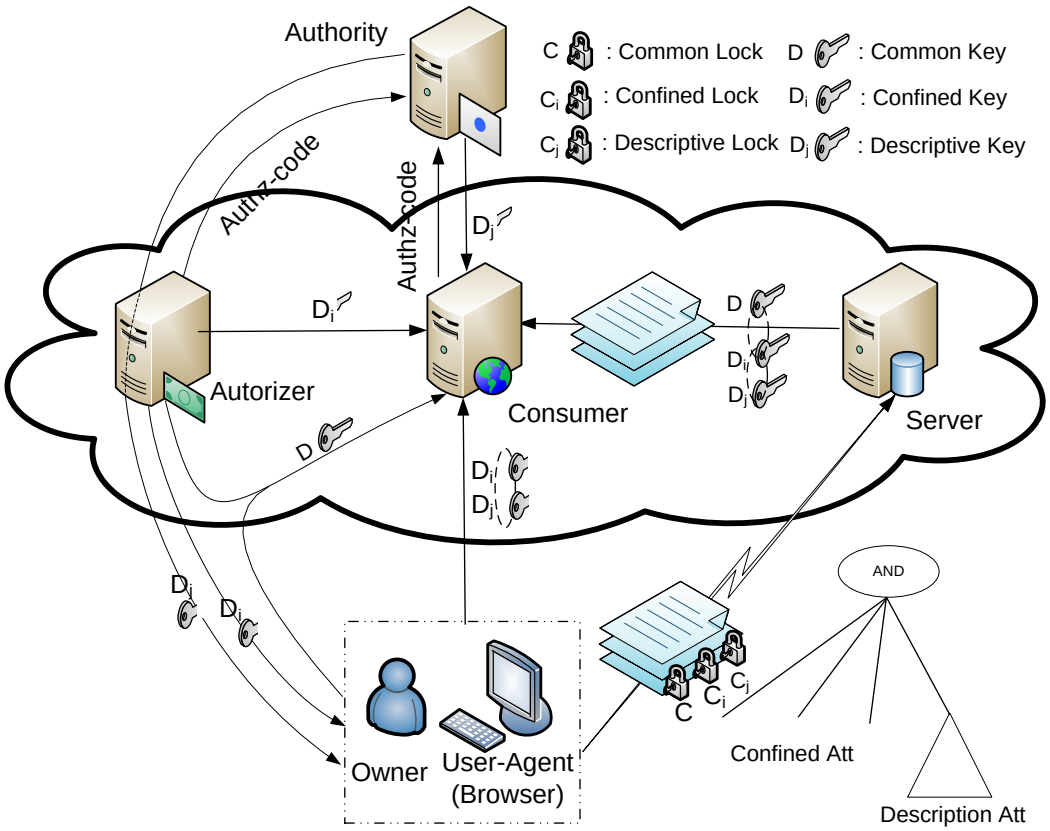

Figure 5.1: The entities and token management in AAuth

Data owners (O): (owners for short) entities, i.e., end-users or software applications, who have resource ownerships and the right to grant access to protected data.

Cloud servers (S): (servers for short) cloud-storage or cloud-database providers that host protected data and provide basic data-services, i.e., read, write, and delete.

Consumers(C): web or traditional-application providers that use owners' data to provide services to the owners.

Authority(AA): trusted organizations or agencies who legitimately define descriptive attributes to eligible consumers.

The authorizer(AZ): the server who runs the AAuth protocol, then issues ABE-based tokens to eligible consumers.

In this system, owners store their sensitive data in cloud servers. When owners ask consumers for services using the owner's data, the owners must grant the authorization to consumers. Unlike OAuth, in which owners, an authorizer, and servers are in the same trusted domain, our model exploits a modified CP-ABE scheme for delegating authorization in semi-trusted environments.

In this semi-trusted environment, owners trust an authorizer and an authority to generate ABE-tokens on the owners' behalves but the owners still contribute to token generation. In addition, the owners trust that cloud servers are honest enough to provide basic data services (i.e., read, write, delete, etc.) but may be curious about owner data or disobey the access policies. To protect data from unauthorized access in semi-trusted clouds, an owner encrypts data with an access policy defined over an attribute set, then stores encrypted data in a server. When a consumer wants to use data, the consumer asks the owner, the authorizer, and the authority jointly to generate an ABE-token. Thus, only the consumer who satisfies the policy can decrypt the encrypted data file. For simplicity, we assume that only owners have read and write privilege on their data, while consumers can read only authorized data. We also assume that without end-users' certificates, the authentication systems, e.g., user-password databases, Active Directory [1]/LDAP [112] [113] , or OpenID [93], are available for authorizers to verify owners. Meanwhile, all service providers, i.e., authorizers, authorities, consumers, and servers, register for public-key certificates from Certificate Authorities (CA) in order to support SSL/TLS [35] security channels.

5.2.2 Adversary Model

In this scenario, our semi-trusted environment means that although no entity trusts the others, everyone trusts the protocol. That is, all entities will follow the proposed protocol in general because protocol violation is easy to detect. However, each entity may exploit some threats to attack the system as follows.

1. We assume that servers can be trusted to provide data-services properly but may be curious about sensitive information and prone to reveal data to ineligible parties.

2. The authorizer may disobey owners' orders to issue tokens, or issue any arbitrary tokens to its conspirators.

3. Consumers may try to get unauthorized files from honest servers by fabricating tokens to obtain unauthorized accesses, resubmitting previous tokens (replay attacks).

4. Without user public-key certificates, owners may propose tokens on behalf of others.

5. Internet users may launch general network attacks on encrypted data or tokens. However, we assume that the communications among CSPs are secure and authentic under SSL/TLS secure channels. Adversaries do not have enough computing power to break cryptographic primitives.

5.2.3 Design Principles

Our design principle is to improve the OAuth standard in order to support inter-operation between servers and consumers in public-cloud environments in which the authorizer and servers are semi-trusted servers. To achieve this goal, we comply with the following design principles.

1. The design should be compatible with the original standard, so it can be implemented as an extension standard.

2. The design should provide end-to-end authorization that does not rely on the policy enforcement in servers.

3. The user-centric approach allows owners to take control of granting access permission, while authorities prove the qualification of designated consumers.

4. The design must achieve data-security goals, i.e., confidentiality and integrity, in this hostile environment by leveraging end-to-end encryption and signing/verification.

5. We also strive for simplicity, efficiency and scalability.

Although our design is defined for the authorization between data owners and application service providers, AAuth is generic enough to apply to other use cases such as user-to-user and provider-to-provider. Based on the above system model, adversary model, and design goals, we present how to construct our scheme and how the protocols manipulate our proposed scheme for the above requirements.

5.3 Definitions and Notations

To construct AAuth, six main security associations (i.e., attributes, access policies, access trees, meta-data, modified CP-ABE, and archive files) are required and described in the following subsections.

5.3.1 Attributes

Our scheme divides the attribute universe into two disjointed sets: a confined set and a descriptive set. To restrict the scope and limit the lifetime of tokens, the confined attributes are mandatory and issued by an authorizer on behalf of owners. The syntax and semantics of confined attributes are defined according to token restrictions, and reserved to disjoin them from descriptive attributes, defined below. Hence, an authorizer publishes the syntax as $\langle attribute \rangle = \langle value \rangle$ and the semantics as follows:

FILE-LOC $= URI$**:** a file identifier consisting of $URL/absolute\ path/filename$;

OWNER $= ownerId$**:** the identifier of a file owner;

PERMIS $= \langle r|w \rangle$**:** file permissions, where 'r' is read only and 'w' is write;

SEC-CLASS $= \langle 1-5 \rangle$**:** a security class of a file, defined in ascending order; and

TIMESLOT $= yyyy/mm/dd/hh/nn$**:** the digits of year, month, date, hour, and minute in the time slot.

Since our ABE-tokens are a set of private keys, rather than issue a token for each file, it is more flexible and efficient to issue a token for multiple files or time slots when the other attributes are shared in common. Furthermore, the granularity of a time slot can be adjusted by masking the fine-grained components with * in ascending order. However, every token of the same file must have the same granularity level. Note that for some cloud storage, FILE-LOC attributes are not physical locations, for instance, objects in a bucket of Amazon S3.

On the other hand, descriptive attributes, which describe consumer characteristics, are defined by authority(s) who monitor and control the consumer. We define the syntax of descriptive attributes as $\langle attribute \rangle @ \langle url \rangle = \langle value \rangle$, where $\langle url \rangle$ is a URL of the authority issuing the attribute. However, authorities freely define the semantics of attributes under their control, then publish them in any public servers.

5.3.2 Access Policies

To encrypt protected data with access policies, we first define the policy in a boolean algebraic expression that combines confined and descriptive attributes together at the root node. Therefore the algebra is constructed by AND each confined-attribute term and the whole set of descriptive-attribute terms is as follows:

$$\begin{aligned} \textit{Policy } \mathbb{A} = & [\text{FILE-LOC}] \text{ AND } [\text{OWNER}] \text{ AND} \\ & [\text{SEC-CLASS}] \text{ AND } [\text{PERMIS}] \text{ AND} \\ & [\text{TIMESLOT}] \text{ AND} \\ & [(\text{OWNER@AUTHZ}) \text{ OR} \\ & (\textit{Descriptive Boolean Algebra})]. \end{aligned}$$

To ignore the descriptive term when an owner accesses his/her own data, a special attribute 'OWNER@AUTHZ' will be OR with the descriptive term. Hence, an owner must request 'OWNER@AUTHZ' from an authorizer when the owner wants a token that does not depend on descriptive attributes.

5.3.3 Access Tree

Basically, in a top-down manner, we can construct an access tree according to a monotonic access policy as in the following algorithm.

1. Staring with the root node, to create t_R-degree polynomial $q_R(\cdot)$, an algorithm sets the point at zero $q_R(0) = s$ for a secret value $s \in \mathbb{Z}_p$ and randomly chooses other $(t_R - 1)$ points.
2. For other nodes x, the algorithm sets $q_x(0) = q_{parent(x)}(index(x))$ and randomly chooses other $(t_x - 1)$ points to define t_x-degree polynomial $q_x(\cdot)$.
3. At each leaf node x, an associated attribute $att(x)$ is assigned to node x.

Here $parent(x)$ denotes the parent node of node x, $att(x)$ denotes the attribute associated with node x if x is a leaf node, $index(x)$ denotes an index number associated with node x, and the index number is assigned uniquely and ascendantly along the tree from the root node ($index = 0$) to the last leaf nodes. As a result, the access policy $\mathbb{A}$, consisting of both confined and descriptive attributes, will convert to an access tree τ, as shown in Figure 5.2.

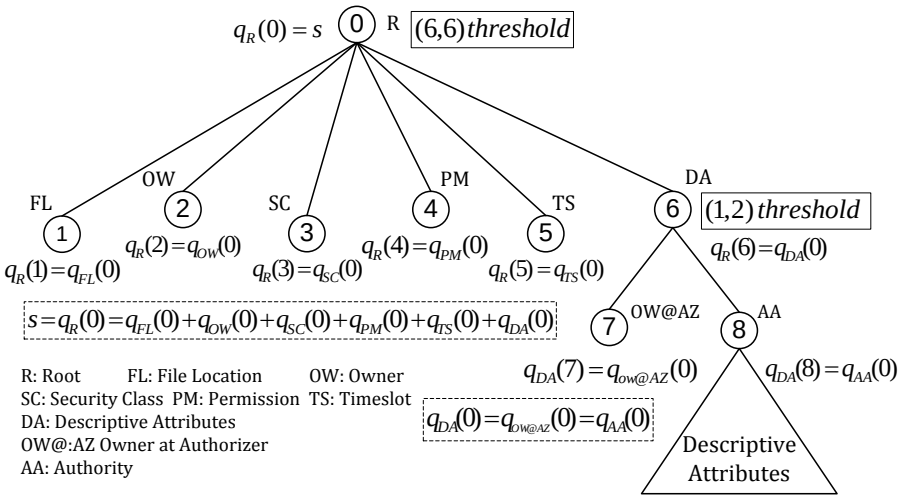

Figure 5.2: Top level of an AAuth access tree

5.3.4 Meta-Data

For user and file management, our scheme maintains meta-data in a user-directory and file-directory that are hosted by an authorizer. A user-directory is the owner list of which IDs and credentials can be maintained by a local database or external identity management center, e.g., LDAP, Active Directory, or OpenID. A file-directory is a repository that an authorizer uses to maintain the last-authorized time slot, last share $q_{TS}(0)$ of the TIMESLOT node, and granularity in each file. This file information is used for time slot synchronization (discussed in Subsection 5.4.7).

5.3.5 Modified CP-ABE

Bethecour *et al.*[14] proposed a CP-ABE construction based on a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, a Linear Secret-Sharing (LSS) scheme for access-tree construction, and a hash function $H : \{0,1\}^* \to \mathbb{G}_1$ that maps any attribute binary-string to a random element in $\mathbb{G}_1$. For adopting their scheme in our distributed scheme, we modify the original scheme in the following five algorithms:

Setup(k). According to a security parameter k, the algorithm chooses a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ of prime order p with generator g of group $\mathbb{G}_1$ and a hash function H :

$\{0,1\}^* \to \mathbb{G}_1$. The authorizer chooses a random exponent β for a Master Private(Secret) Key $MSK = \langle \beta \rangle$ and publishes a Master Public Key

$$MPK = \langle \mathbb{G}_1, g, h = g^{\beta}, f = g^{1/\beta} \rangle.$$

Meanwhile, each owner chooses a random exponent α for an Owner Private(Secret) Key $OSK = \langle g^{\alpha} \rangle$ and publishes an Owner Public Key

$$OPK = \langle e(g,g)^{\alpha} \rangle.$$

in the user-directory of the authorizer.

$Encrypt(MPK, OPK, m, \mathbb{A})$. This algorithm encrypts a message m under the access tree τ. An owner first chooses a random value $s \in \mathbb{Z}_p$, then constructs an access tree τ according to $q_R(0) = s$ and an access policy $\mathbb{A}$. Let Y be the set of leaf nodes in τ. Then a ciphertext CT is computed by

$$CT = \langle \tau, \tilde{C} = m \cdot e(g,g)^{\alpha s}, C = h^s, \forall y \in Y \ : \ C_y = g^{q_y(0)}, C'_y = H(att(y))^{q_y(0)} \rangle.$$

$KeyGen(MSK, OSK, \omega)$. The algorithm will take as input a set ω of attributes. The authorizer chooses a random value $r \in \mathbb{Z}_p$ and random values $r_i \in \mathbb{Z}_p$ for each confined attribute $i \in \omega'$; an authority chooses random values $r_j \in \mathbb{Z}_p$ for each descriptive attribute $j \in \omega''$; and an owner chooses a random value a.

Let $\omega = \omega' \cup \omega''$ denotes the attribute set and $k \in \omega$ denotes each attribute. With an ElGamal-like mask, the authority, the authority, and the owner jointly compute a user Private(Secret) Key SK without the revealing their own private keys to each other as follows:

$$SK = \langle D = g^{(\alpha + ra)/\beta}, \forall k \in \omega \ : \ D_k = g^{ra} H(k)^{r_k}, D'_k = g^{r_k} \rangle.$$

$Delegate(SK, \tilde{\omega})$. The algorithm takes as input a private key SK, which is for an attribute set ω and another attribute set $\tilde{\omega} \supseteq \omega$. The algorithm first chooses a random value $\tilde{r}$ and $\{\tilde{r}_l \ \mid \forall l \in \tilde{\omega}\}$. Then it creates a new private key $\widetilde{SK}$ for an attribute set $\tilde{\omega}$ as

$$\widetilde{SK} = \langle \tilde{D} = D \cdot f^{\tilde{r}}, \forall l \in \tilde{\omega} \ : \ \tilde{D}_l = D_l \cdot g^{\tilde{r}} \cdot H(l)^{\tilde{r}_l}, \tilde{D}'_l = D'_l \cdot g^{\tilde{r}_l} \rangle.$$

$Decrypt(CT, SK)$ This algorithm is a recursive algorithm over the access tree τ for a ciphertext CT. Let $DecryptNode(CT, SK, x)$ denotes a node algorithm, which takes as input a ciphertext CT, a private key SK, and a node x in τ; and let $k = att(x)$. If x is a leaf node and $k \in \omega$, then

$$DecryptNode(CT, SK, x) = \frac{e(D_k, C_x)}{e(D'_k, C'_x)} = \frac{e(g^{ra} \cdot H(k)^{r_k}, g^{q_x(0)})}{e(g^{r_k}, H(k)^{q_x(0)})} = e(g, g)^{raq_x(0)}.$$

Otherwise, we define $DecryptNode(CT, SK, x) = \perp$.

When x is a non-leaf node, the node algorithm proceeds in a recursive fashion as follows: for each child node z of x, z calls $DecryptNode(CT, SK, z)$ and stores the output as F_z. Let ω_x be an arbitrary t_x-sized set of x's child nodes such that $F_z \neq \perp$ and t_x is the threshold value of the threshold gate at node x. If there exits such a set, then let $k = index(z)$, $\hat{\omega}_x = \{index(z) \mid z \in \omega_x\}$, and compute

$$\begin{aligned}
F_x &= DecryptNode(CT, SK, x) \\
&= \prod_{z \in \omega_x} F_z^{\triangle_{k,\hat{\omega}_x}(0)} \\
&= \prod_{z \in \omega_x} (e(g, g)^{raq_z(0)})^{\triangle_{k,\hat{\omega}_x}(0)} \\
&= \prod_{z \in \omega_x} ((e(g, g)^{raq_{parent(z)}(index(z))}))^{\triangle_{k,\hat{\omega}_x}(0)} \qquad \text{(by construction)} \\
&= \prod_{z \in \omega_x} ((e(g, g)^{raq_x(k)\triangle_{k,\hat{\omega}_x}(0)} \\
&= e(g, g)^{raq_x(0)} \qquad \text{(by interpolation).}
\end{aligned}$$

Otherwise, we define $DecryptNode(CT, SK, x) = \perp$.

From the node algorithm, $Decrypt(CT, SK)$ can be computed by calling the node algorithm from the root node R of the access tree τ. If the access tree τ is satisfied by ω, then we have

$$A = DecryptNode(CT, SK, R) = e(g, g)^{raq_R(0)} = e(g, g)^{ras} \qquad \text{(by interpolation).}$$

Now the decryption can be computed by

$$Decrypt(CT, SK) = \tilde{C}/(e(C, D)/A) = \tilde{C}/(e(h^s, g^{(\alpha+ra)/\beta})/e(g, g)^{ras}) = m.$$

5.3.6 Archive File

Although we can directly use an ABE scheme to encrypt protected data, our scheme separates encryption into two levels: header encryption and data encryption. This separation can improve efficiency because 1) asymmetric-key encryption itself is not as efficient as symmetric-key encryption, 2) our policy change and time slot synchronization do not necessitate data re-encryption, so it can be delayed until the data is changed. To this end, we encapsulate protected data in an archive file consisting of a header encrypted with an ABE key and the protected data encrypted with a symmetric encryption.

$$\langle Archive \rangle = \{\langle Header \rangle\}_{ABE} \parallel \{\langle Data \rangle\}_{KE} \parallel \langle \mathbb{A} \rangle \parallel \langle IntegTag \rangle,$$

Next, we define the parameters in a header as follows:

$$\langle Header \rangle = \langle FileDesc \rangle \parallel \langle EncryMeth \rangle \parallel \langle IntegMeth \rangle \parallel \langle KE \rangle \parallel \langle KV \rangle \parallel \langle \mathbb{A} \rangle$$

$\langle FileDesc \rangle$**:** the description of protected-file content.

$\langle EncryMeth \rangle$**:** a symmetric-key algorithm is used to encrypt protected data, such as AES-128, AES-192, RSA-1024, RSA-2048, etc.

$\langle IntegMeth \rangle$**:** a set of algorithms is used to generate an integrity tag, such as RSA-MD5, RSA-SHA1, DSA-MD5, DSA-SHA1, etc.

$\langle KE \rangle$**:** a symmetric key is used to encrypt protected data.

$\langle KV \rangle$**:** an asymmetric key is used to verify an integrity tag.

$\langle \mathbb{A} \rangle$**:** an access policy is recorded in plaintext form.

$\langle IntegTag \rangle$**:** an integrity tag is generated from clear $\langle Header \rangle$ and encrypted data.

5.3.7 Notation

Note that the notation introduced in this section and shown in Table 5.1 is used throughout this chapter. In addition to all components in this section, our scheme requires procedures and protocols described in the next section to provide security in semi-trusted clouds.

Table 5.1: Notation used in AAuth

Notation	Description
MPK, MSK	System public and private keys of an authorizer
OPK, OSK	System public and private keys of owners
Att_x, PK_x	Attribute x, user public key for PK_x
SK	User private key consisting of two parts: common D and {part1 D_x, part2 D'_x} for each Att_x
D_i, D'_i	Part1 and part2 key components for confined Att_i
D_j, D'_j	Part1 and part2 key components for descriptive Att_j
$\hat{D}_i, \hat{D}_j$	The partial part1 of confined and descriptive key components D_i, D_j
$\{U\}_X$	The cyphertext of a string U encrypted with X's public key
$[U]_X$	A string U attached with digital signature over a string U generated with X's private key

5.4 AAuth Procedures and Protocols

Our scheme extends OAuth to a cryptographic token system in which its ABE-token is a private key associated with a set of attributes, and its protected resource (data file) is encrypted with an access policy constructed from an access structure over a public key. Due to limitations of lifetime and scope in tokens and the necessity of multi authorities, our scheme divides the attribute universe in two disjointed sets: confined attributes defined by owners to limit the lifetime and scope of tokens, and descriptive attributes defined by authority(s) to certify the characteristics of consumers. To allow owners to contribute to token generation, we separate the master key of CP-ABE into two parts: g^α for owners and β for a authorizer, then add another level of ElGamal-like masks to conceal the master keys from each other during key generation.

AAuth mainly requires three off-line procedures: setup, file encapsulation, and file decapsulation, and four on-line protocols: service request, token request, file access, and time slot synchronization. Optionally, AAuth can provide key delegation, policy changes, and data updates. This section explains these procedures and protocols. Note that resource servers have no affiliation in authorization but provide basic data-services: read, write, delete.

5.4.1 Setup Procedure

Before AAuth starts to provide inter-operations, an authorizer and owners must initialize the system parameters in Procedure 1.

Procedure 1. Setup phase

1. An authorizer chooses a security parameter k and runs the CP-ABE algorithm $Setup(k)$ that outputs a bilinear group $\mathbb{G}_1, \mathbb{G}_2$, a bilinear map e, a generator g of $\mathbb{G}_1$, and a hash function H. All outputs in this step are published in a public server.

2. The authorizer chooses a random value $\beta \in \mathbb{Z}_p$ and keeps it secretly, then generates a master public key $MPK = \langle \mathbb{G}_1, g, h = g^{\beta}, f = g^{1/\beta} \rangle$ and publishes it in a public server.

3. Each owner contributes to key generation by randomly selecting $\alpha \in \mathbb{Z}_p$, generates an owner public key $OPK = e(g, g)^{\alpha}$ that is published in a user-directory of the authorizer (not really published), then keeps an owner private key $OSK = g^{\alpha}$ secretly.

Our explanation for the scheme is based on the following example from the OAuth standard:

Example 1. Jane (an owner) has recently uploaded some private photos (protected resources) to her sharing site 'photos.com' (a server). She would like to use the 'printer.com' website (a consumer) that is certified by a trusted authority (authority.org), to print her photos. Jane does not have a public-key certificate and does not wish to share her identity and credentials with 'print.com'. However, she has registered with a trusted mail provider (mail.net) that also provides OAuth services for its subscribers.

5.4.2 File Encapsulation Procedure

Before uploading files to a cloud server, an owner encrypts and encapsulates data files into archive files by using Procedure 2.

Procedure 2. File encapsulation phase

1. Define an access policy $\mathbb{A}$ from both confined and descriptive attributes as follows:

```
# Confined attributes
[FILE-LOC=http://photos.com/2010/brunce/pic-1]
AND   [OWNER=Jane@photos.net]
AND   [SEC-CLASS=3]
AND   [PERMIS=r]
AND   [TIMESLOT=2011/06/27/13/**]
AND   # Descriptive attributes
      [(OWNERe@mail.net=Jane@mail.net) OR
      [(NAME@authority.org=printer.com) AND
       (SERVICE@authority.org = print) AND
       (LOCAT@authority.org = canada) OR
       (TRUST-LEV@authority.org = 3)]].
```

Then convert $\mathbb{A}$ to an access tree τ by using the algorithm in Subsection 5.3.3.

2. Randomly choose an encryption key KE, and generate signature key KS and verification key KV, then define all parameters in the header H according to the format in Subsection 5.3.6.

3. Encrypt a data file with the key KE, and generate an integrity tag from the encrypted data-file and the clear header with the key KS.

4. Encrypt the header H by choosing a random value s, then using the CP-ABE algorithm $Encrypt(MSK, m, \tau)$ to compute a ciphertext CT.

5. Construct an archive file from the encrypted header and encrypted data file, the access policy, and the integrity tag in the format described in Subsection 5.3.6. Then the owner stores the archive file in the cloud server using a regular API.

Note that we compute an integrity tag from an unencrypted header to avoid the effect of time slot synchronization (described in Subsection 5.4.7). We have now finished describing how to create and store ciphertext; next we show how to obtain an ABE token and how to retrieve and decrypt the ciphertext.

5.4.3 Service Request Protocol

In this scenario, owners first ask consumers for services that require the owners' data. Before a consumer can access data, it must ask an authorizer for ABE-tokens issued on behalf

of owners. Then the consumer uses a private key in a token to prove the authorization by performing a challenge-response protocol with a cloud server. If the verification succeeds, the server will provide the archive file according to the token. So our scheme starts from an owner asking a service (e.g., to print photos) that require the owner's data files from a consumer, as described in Protocol 1 and shown in Figure 5.3. Note that owners registered with an authorizer and consumers registered with an authority are beyond the scope of this work.

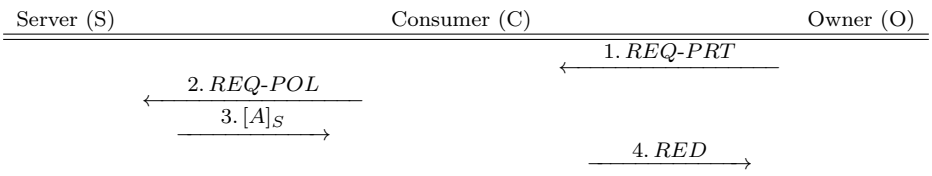

Figure 5.3: The message flow of protocol 1: service request

Protocol 1. Service request phase

1. $O \rightarrow C$: First an owner sends a command $REQ\text{-}PRT$ to request a printing service from the consumer 'printer.com'.

2. $C \rightarrow S$: Since the service requires the owner's data, the consumer sends a server a file location with a command $REQ\text{-}POL(FileLoc)$ to request the required file's access policy from the server 'photos.com'.

3. $S \rightarrow C$: The server extracts the access policy '$\mathbb{A}$' from the archive file and signs $\mathbb{A}$. Then the signed access-policy $[\mathbb{A}]_S$ is replied to the consumer.

4. $C \rightarrow O$: To initiate a token request, the consumer sends the owner an HTTPS-redirect command RED to redirect the owner's user-agent to the authorization page 'https://authorizer.net/authorize' on an authorizer. To this end, the consumer ID and a redirection URI ($https://printer.com/ready$) signed by the consumer $[ID_C, RED\text{-}URI]_C$ and $[\mathbb{A}]_S$ received earlier are included in the command as $RED([ID_C, RED\text{-}URI]_C, [\mathbb{A}]_S)$.

5.4.4 Token Request Protocol

Once an owner's user-agent is redirected to an authorizer, the authorizer first verifies the owner's ID and credentials. If the authentication succeeds, the owner and the authorizer

cooperate to issue an ABE-token for the consumer, as described in Protocol 2 and shown in Figure 5.4.

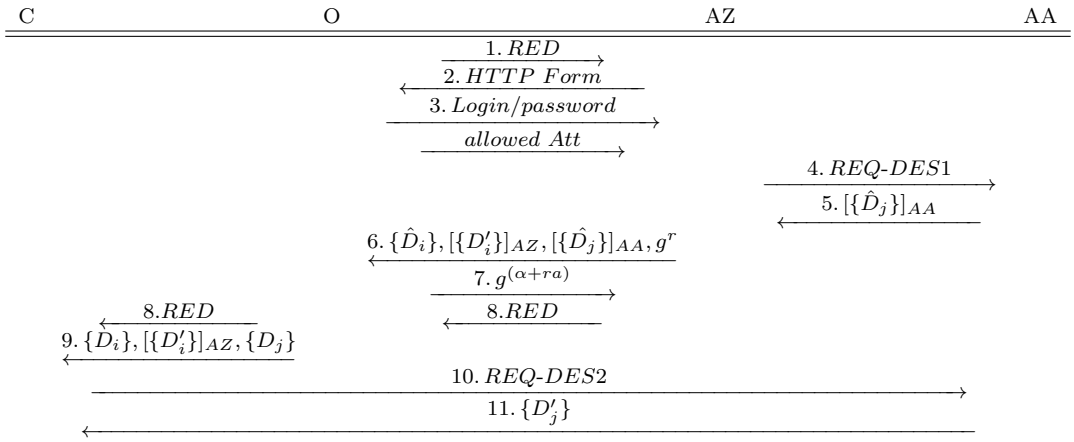

Figure 5.4: The message flow of protocol 2: token request

Protocol 2. Token request phase

1. $O \rightarrow AZ$: From the redirect command, an owner's browser will pass $[ID_C, RED\text{-}URI]_C, [\mathbb{A}]_S$ to an authorizer.

2. $AZ \rightarrow O$: The authorizer thus uses HTTP form pages to authenticate the owner and receive the owner's decision, i.e, allowed confined attributes.

3. $O \rightarrow AZ$: The owner sends her ID and credentials to authenticate herself to the authorizer. Then the owner also defines confined attributes that she allows and sends these attributes to the authorizer: For example,

```
FILE-LOC=http://photos.com/2010/brunce/pic-1,
FILE-LOC=http://photos.com/2010/brunce/pic-2,
OWNER=Jane@mail.net, PERMIT=r, SEC-CLASS=3;
/* current time slot */
TIMESLOT=2011$|$06$|$27$|$13$|$**,
/* future time slot(s)*/
TIMESLOT=2011$|$06$|$27$|$14$|$**.
```

4. $AZ \rightarrow AA$: If the authentication succeeds, the authorizer generates an authorization code *Autz-code* (a nonce) and sends the authority a command $REQ\text{-}DES1(ID_C, Authz\text{-}Code)$ to request descriptive components.

5. $AA \rightarrow AZ$: The authority will retrieve the consumer's attributes; for example,

   ```
   NAME@attribute.org = printer.com
   SERVICE@attribute.org = print,
   LOCAT@attribute.org = canada,
   TRUST-LEV@attribute.org = 2.
   ```

   Then the authority generates and signs the partial part-1 $[\{\hat{D}_j\} = \{H(j)^{r_j}\}]_{AA}$ of descriptive components, and replies it to the authorizer, while the part-2 $\{D'_j\} = \{g^{r_j}\}$ will be directly sent to the consumer in the file access protocol (described in Subsection 5.4.5).

6. $AZ \rightarrow O$: The authorizer generates the partial part-1 $\{\hat{D}_i\} = \{H(i)^{r_i}\}$ and part-2 $\{D'_i\} = \{g^{r_i}\}$ according to the owner's decision, and randomly selects $r \in \mathbb{Z}_p$. Then the partial part-1 $\{\hat{D}_i\}$ and the part-2 $[\{D'_i\}]_{AZ}$ which are signed by the authorizer, g^r, and the descriptive part-1 $[\{\hat{D}_j\}]_{AA}$ received earlier are sent to the owner.

7. $O \rightarrow AZ$: The owner verifies whether the confined components are associated with the attributes by computing bilinear pairing $e(\hat{D}_i, g) \stackrel{?}{=} e(D'_i, H(i))$. If the verification succeeds, the owner randomly chooses a, computes $g^{\alpha+ra}$ from g^r, a and her private key $OSK = g^\alpha$, then replies the result $g^{\alpha+ra}$ to the authorizer.

8. $AZ \rightarrow O$: Now the authorizer is ready to generate the common part D from the MSK β and $g^{\alpha+ra}$ received earlier. In this cooperation with ElGamal-like masking, the authorizer only knows $g^{(\alpha+ra)}$, and the owner only knows g^{ra}. The authorizer encrypts the common part $D = g^{(\alpha+ra)/\beta}$ and the authorization code *Authz-code* with the consumer's public key and signs with the authorizer's private key, before sending the owner a redirect command $RED([\{g^{(\alpha+ra)/\beta}, Authz\text{-}Code\}_C]_{AA})$ to redirect the user-agent back to the consumer at the redirection URI endpoint.

9. $O \rightarrow C$: The owner binds both the partial part-1 $\{\hat{D}_i\}, \{\hat{D}_j\}$ of confined and descriptive components by multiplying them with g^{ra} and sends all keys $\{D_i = g^{ra}H(i)^{r_i}\}$, $[\{g^{r_i}\}]_{AZ}$, $\{D'_j = g^{ra}H(j)^{r_j}\}$ to the consumer.

10. $C \rightarrow AA$: The consumer sends the authority a command $REQ\text{-}DES2(ID_C, Authz\text{-}Code)$ to authenticate itself and to request the descriptive part-2.

11. $AA \rightarrow C$: If the authentication succeeds, the authority will reply to the consumer with the part-2 $\{D'_j = g^{r_j}\}$ of the descriptive components.

Now the consumer holds an ABE token and is ready to access data files, which will be illustrated in the next subsection.

5.4.5 File Access Protocol

After a consumer obtains an ABE token, the consumer must prove private-key possession by performing a challenge-response with a cloud server as describe in Protocol 3 and shown in Figure 5.5.

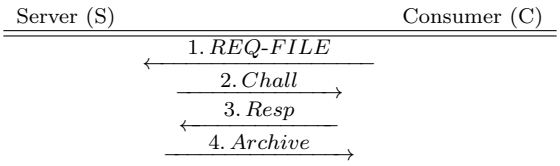

Figure 5.5: The message flow of protocol 3: file access

Protocol 3. File access phase

1. $C \rightarrow S$: First a consumer sends a server a command $REQ\text{-}FILE(FileLoc)$ that includes a file location to request a challenge from the server.

2. $S \rightarrow C$: The server generates a challenge value $chall = Encrypt(MPK, OPK, nonce, \mathbb{A} \backslash Att_{TS})$ from a nonce encrypted by CP-ABE encryption with the access policy, excluding the time slot attribute, then sends the challenge to the consumer.

3. $C \rightarrow S$: The consumer now has a complete ABE private key that satisfies the access policy. Hence, the consumer can generate a response $Resp = Decrypt(Chall, SK)$ by using the ABE decryption algorithm and replies the server with the response $Resp$.

4. $S \rightarrow C$: The server verifies the challenge-response by $Resp \stackrel{?}{=} nonce$. If it succeeds, the server replies the consumer with the archive-file $Archive$.

Now the consumer has both the archive file and the ABE private key, so the next step is data verification and decryption of an archive file.

5.4.6 File Decapsulation Procedure

Before file decryption, a consumer verifies the integrity of the archive file. Thus, a consumer performs decapsulation of an archive file as in Procedure 3.

Procedure 3. File decapsulation phase

1. To extract all parameters from the header, the consumer decrypts the header by using the CP-ABE algorithm $Decrypt(CT, SK)$.
2. The data integrity of the decrypted header and encrypted data must be checked by the algorithm and key, which are defined in $IntegMeth$ and KV. Then the consumer moves to the next step if the integrity is valid.
3. The $FDesc$ and $\mathbb{A}$ parameters are used to check whether the archive file is the correct one.
4. The encrypted data file is decrypted by the algorithm and key, which are defined in $CrypMeth$ and KE.

Using all the above procedures, the consumer can access the plaintext of a data file. In addition, he can verify the integrity and correctness of the data. In the remaining subsections, we present other procedures that facilitate our scheme.

5.4.7 Time Slot Synchronization Protocol

In AAuth, the access permission in each time slot is granted by matching between the key component $D_{ts} = g^r H(ts)^{r_{ts}}$ in SK and the ciphertext component $H(ts)^{q_{TS}(0)}$ in the access tree (see Subsection 5.3.3) of a ciphertext. Hence, AAuth controls the lifetime of tokens by defining the key component(s) of the time slot attribute in a token, and automatically re-encrypting the header of a file with the current time slot attribute. As a result, a consumer can access a file only in the time slot(s) defined in the token.

To this end, an owner (encrypter) must send the share $q_{TS}(0)$ of the time slot attribute to an authorizer when performing file encapsulation. For each time slot, the authorizer will maintain the last time slot share $q_{TS}(0)$ of each file in its file directory in order to compute two ciphertext components C_y, C'_y and two update values $h^{\tilde{s}}, e(g,g)^{\alpha\tilde{s}}$. These components and values will be sent to a cloud server to update ciphertext components and re-mask the header. Thus, the previous mask $e(g,g)^{\alpha s}$ which other consumers (decrypters)

have already occupied will be disabled. Note that re-masking a header does not affect an integrity tag because the tag is computed from the unencrypted header. Updating ciphertext components and re-masking a header must be performed by cooperation between an authorizer and a server in Protocol 4 in real-time.

Protocol 4. Time slot synchronization phase

1. In each time slot t, an authorizer chooses a new random value $\tilde{s}(t)$, computes a new share $q_{TS}(0,t) = q_{TS}(0,t-1) + \tilde{s}(t)$ of a TIMESLOT node, then saves the new share as the last share in the file directory.

2. New ciphertext components for a new time slot can be computed by $C_{TS}(t) = g^{q_{TS}(0,t)}, C'_{TS}(t) = H(Att_{TS}(t))^{q_{TS}(0,t)}$, where $Att_{TS}(t)$ is the string of t-th time slot.

3. Two update values $h^{\tilde{s}(t)}$ and $e(g,g)^{\alpha\tilde{s}(t)}$ are computed from $MPK = g^{\beta}, OPK = e(g,g)^{\alpha}$.

4. Then the ciphertext components and update values are sent to a server.

5. The server replaces two ciphertext components C_{TS}, C'_{TS} with the received components according to the current time slot.

6. The server also updates the value $C(t) = C(t-1) \cdot h^{\tilde{s}(t)} = h^{s(t-1)+\tilde{s}(t)}$ and re-masks the header $\tilde{C}(t) = \tilde{C}(t-1) \cdot e(g,g)^{\alpha\tilde{s}(t)} = m \cdot e(g,g)^{\alpha(s(t-1)+\tilde{s}(t))}$ in a ciphertext.

In the above procedure, we obviously retain the consistency between the TIMESLOT share in the access tree and the secret value masking the header. That is, in each time slot t, its share $q_{TS}(0,t)$ can be combined with the other shares to construct the corresponding secret value $s(t)$. This fact results from the root node in our access tree being an AND gate, i.e., an (n,n) threshold gate, which causes $s = q_{FL}(0) + q_{OW}(0) + q_{SC}(0) + q_{PM}(0) + q_{TS}(0) + q_{DA}(0)$ (see Figure 5.2). Then the protocol adds two sides of the equation with a new random value $\tilde{s}(t)$. Note that the TIMESLOT attribute in a header and a tail is a time stamp when the protected file was encapsulated, not the time slot that encrypted the header. Figure 5.6 shows the propagation of TIMESLOT shares, ciphertext components, and associated secret masks, according to each time slot.

Note that the above real-time synchronization has a tight time synchronization similar to that of the Kerberos system. Next, we propose loose synchronization that can cope with such a problem and optimize both computation and communication costs in two approaches: sender-push and receiver-pull [39].

Timeslot	0	1	$\cdots$	$n-1$	n
Random value, $\tilde{s}$	$\tilde{s}(0)$	$\tilde{s}(1)$	$\cdots$	$\tilde{s}(n-1)$	$\tilde{s}(n)$
Share, $q_{TS}(0)$	$q_{TS}(0,0)$	$q_{TS}(0,1) = q_{TS}(0,0) + \tilde{s}(1)$	$\cdots$	$q_{TS}(0,n-1)$	$q_{TS}(0,n) = q_{TS}(0,n-1) + \tilde{s}(n)$
Component, C_{ST}	$C_{ST}(0)$	$C_{ST}(1) = g^{q_{TS}(0,1)}$	$\cdots$	$C_{ST}(n-1)$	$C_{ST}(n) = g^{q_{TS}(0,n)}$
Component, C'_{ST}	$C_{ST}(0)$	$C'_{ST}(1) = H(Att_{ST}(1))^{q_{TS}(0,1)}$	$\cdots$	$C_{ST}(n-1)$	$C'_{ST}(n) = H(Att_{ST}(n))^{q_{TS}(0,n)}$
Component, C	$C(0)$	$C(1) = C(0) \cdot h^{\tilde{s}(1)}$	$\cdots$	$C(n-1)$	$C(n) = C(n-1) \cdot h^{\tilde{s}(n)}$
Component, $\tilde{C}$	$\tilde{C}(0)$	$\tilde{C}(1) = \tilde{C}(0) \cdot e(g,g)^{\alpha\tilde{s}(1)}$	$\cdots$	$\tilde{C}(n-1)$	$\tilde{C}(n) = \tilde{C}(n-1) \cdot e(g,g)^{\alpha\tilde{s}(n)}$
Secret mask, s	$s(0)$	$s(1) = s(0) + \tilde{s}(1)$	$\cdots$	$s(n-1)$	$s(n) = s(n-1) + \tilde{s}(n)$

Figure 5.6: Ciphertext components propagation according to time slot changes

Sender-Push Mode

In this mode, an authorizer completely controls what synchronization data is delivered and when it is delivered, while a cloud server replaces and computes what it receives. Since an authorizer participates in token issues, the authorizer knows which time slots are authorized for each file. With this knowledge, the authorizer will not continue synchronization if no other authorization occurs. For example, if TIMESLOT=2011|06|27|13|** and TIMESLOT=2011|06|27|14|** are authorized, then only the synchronization data, i.e., ciphertext components and update values of time slot 2011|06|27|13|**, 2011|06|27|14|**, and 2011|06|27|15|** must be sent out. Furthermore, the authorizer can deliver multiple time slots at the same time by combining the components and values of all time slots into one message sent to a cloud server. On the other side, the cloud server can reduce the computing cost by caching the synchronization data received from the authorizer until a file access occurs. From the synchronization data in the cache, the server can aggregate update values by multiplying all update values and selecting the last received components for time slot synchronization.

As a consequence, both the authorizer and a cloud server can optimize both computation and communication cost by leveraging two cryptographic primitives: proxy re-encryption [15] and lazy re-encryption [65]. Another advantage is that the authorizer has no responsibility to store and manage such synchronization data until a cloud server is ready to receive them. The disadvantage is that a cloud server must maintain unused synchronization data until it expires.

Receiver-Pull Mode

Another optimization approach is for a cloud server to pull synchronization data from the authorizer when a data file is requested. In this way, the computation and communication cost is minimal because the work load occurs only when a consumer requests a file. Moreover, the authorizer can consolidate synchronization data, i.e., by multiplying update values and selecting the last components, before sending the data to a cloud server. Unfortunately, these optimizations impose two disadvantages. One is that the authorizer cannot control when time synchronization is performed, and the other is that it has to store and maintain ciphertext components and update values until they expire or a cloud server requests them. This model also causes more latency time due to waits for authorizer responses.

In summary, all three approaches, i.e., tight synchronization, sender-push, and receiver-pull, require trade offs between performance and security levels.

5.4.8 Token Delegation Procedure

In some situations, a consumer may ask another provider to process a data file for which authorization already exists. For example, the web site 'printer.example.com' has already obtained a token with two time slots and two files, e.g.,

```
FILE-LOC=http://photos.com/2010/brunce/pic-1,
FILE-LOC=http://photos.com/2010/brunce/pic-2,
SEC-CLASS=3, PERMIS=r,
/* current time slot */
TIMESLOT=2011$|$06$|$27$|$13$|$**,
/* future time slot(s)*/
TIMESLOT=2011$|$06$|$27$|$14$|$**.
```

Using the key delegation algorithm of CP-ABE, the web site 'printer.com' can ask the website 'poster.com' to print a poster for a file 'pic-1' in the time slot '2011|06|27|13|**' by generating a new private-key set associated with

```
FILE-LOC=http://photos.com/2010/brunce/pic-1,
SEC-CLASS=3, PERMIS=r,
/* current time slot */
TIMESLOT=2011$|$06$|$27$|$13$|$**.
```

5.4.9 Policy Changing Procedure

Another reason that we re-encrypt the header of an archive file is to change the access policy. Regardless of confined or descriptive attributes, a naive way is that an owner rebuilds a new access policy, re-encrypts a header, recomputes an integrity tag, and then asks a server to replace all the headers and tails in the archive file. An advantage we can obtain is that there is no need to re-encrypt the protected data.

5.4.10 Data Updating Procedure

Updating data is a straightforward procedure. An owner re-encrypts protected data with the encryption key KE in the header of an archive file, recomputes an integrity tag, and then asks a server to replace the encrypted data and the integrity tag. If both the data and access policy are changed, an owner repeats all steps to rebuilt a new archive file and stores it in a server. However, it is obvious that we can provide write permission to consumers by separating a header into two parts: read and write. The read part is retained in the same format, while the write part includes a signing key and is encrypted by CP-ABE encryption with a write policy.

5.5 Security Analysis

We analyze our scheme from two perspectives: internal and external adversaries. For internal adversaries, all entities in the system are considered as semi-trusted entities, in the sense that they can exploit threats to subvert authorization control and data security, but still honestly follow the protocol. For external adversaries, they may not run the protocol but instead try to launch general attacks to violate data security. To analyze security from internal attacks, we consider the following attacks on AAuth protocol that each entity can launch in the system.

5.5.1 Action of Cloud Server

We consider that cloud servers host sensitive information and are curious about the data or may reveal information to unauthorized consumers. As the data is stored in ciphertext with an integrity tag, neither cloud servers nor unauthorized consumers can decrypt the data without decryption keys or fabricate/modify the data without signature keys. Thus,

data confidentiality and integrity are secure even if the cloud server is compromised. Our authorization is performed in an end-to-end manner, i.e., an access policy is bundled into a ciphertext by an owner and a consumer must then use a key that satisfies the access policy to decrypt. Hence, the access policy is enforced by the decryption algorithm, not by cloud servers, thereby achieving data confidentiality and integrity.

5.5.2 Action of an Authorizer

In the original Kerberos and OAuth, as well as in cloud servers, an authorizer is another dominating entity since it can issue tokens to anyone without owners' permission. This situation is rational if an authorizer is on an owner's premises, not in a public cloud. Our scheme enables secure authorization in public clouds by allowing owners to contribute to token generation and assign confined attributes. Thus, an authorizer alone cannot generate the common part of SK ($D = g^{(\alpha+ra)/\beta}$) for unauthorized consumers because the authorizer has no knowledge about the owner's SK ($OSK = g^{\alpha}$). In addition, an authorizer cannot arbitrarily generate an ABE token because a confined SK proposed by an owner can be verified by the owner.

5.5.3 Action of Owners

Although an owner is unlikely to counterfeit the tokens she proposes, she may ask an authority to generate a token with 'OWNER' attributed in other people's name to access files not belonging to her. In other words, the owner pretends to be someone else. In this case, an authorizer can easily detect this misbehavior since the owner must be authenticated to an authorizer. Also an owner may fabricate the part-1 of confined or descriptive components ($H(i)^{r_i}, H(j)^{r_j}$) and combine it with her combining term g^{ra}. In our scheme, the part-2 of confined component g^{r_i} is signed by an authorizer, and an authority directly sends the part-2 of confined component g^{r_j} to a consumer. Thus, the owner must compute r_i or r_j from g^{r_i} or g^{r_j} respectively. This problem can be reduced to a Discrete Logarithm Problem (DLP), and so fabrication is unsuccessful.

5.5.4 Action of Consumers

Consumers may modify their own tokens; for example, they may change the time slot components in their own tokens. Hence, consumers must try to select the confined components $g^{ra}H(i)^{r_i}$, g^{r_i} for the time slot attribute that can satisfy the policy as the correct key

components, this problem can be reduced to a pairing inversion problem thereby failing on modification. Our scheme can also resist collusion attacks as does the original CP-ABE scheme because in each authorization, an authority and an owner randomly choose new random values r, a respectively to combine confined and descriptive components. Therefore, one consumer cannot combine her keys to create more powerful keys, and multiple consumers cannot collude in combining the different keys to create a new key.

5.5.5 Resistance to Other Attacks

In addition, we consider general attacks from external adversaries, such as eavesdropping, Man-In-The-Middle (MITM), and Denial-of-Service (DoS), etc.

Eavesdropping and active attacks: Data files are encrypted and signed by owners, then verified and decrypted by consumers, thereby performing all cryptographic operations end-to-end. Therefore, eavesdropping on data files cannot disclose data confidentiality, and active attacks cannot corrupt data integrity. Since the header parameters can be verified from the tag, these parameters can be trusted to check the properties, i.e., owner, location, and security class of the data file to verify the correctness of data properties.

MITM attacks: All service providers and authorities in the system register with CA for public-key certificates. Hence, the communication in our scheme can be protected by SSL/TLS channels. Therefore, our scheme can resist MITM attacks. Moreover, the part-2 of confined and descriptive SK (D'_i, D'_j) originate from an authorizer and authority(s) respectively. These two parts are sent over different SSL/TLS channels and combined by a consumer, so an adversary must intercept two SSL/TLS sessions at the same time to obtain a complete SK; this situation rarely happens in practice.

Off-line attacks: Our authorization is divided into two levels: token request and file access. In token request, a consumer must get permission from an owner and qualify with respect to the policy in order to get a satisfactory ABE-token. The consumer then uses the ABE-token to generate a response value for file access. To obtain data files from a cloud server, a consumer must prove his/her authorization by sending a response value satisfying the challenge value. Thus, adversaries cannot obtain ciphertext if they have no authorization, so our scheme can protect encrypted data from off-line attacks.

Credential protection: Like OAuth, an owner can grant assess permission to a consumer, even though the owner does not expose her credential, i.e., password, certificate, Information Card, etc., to the consumer.

Certified entities: An authority is a certifier who describes consumers according to their characteristics by issuing descriptive attributes for eligible consumers. Thus, consumers cannot declare forged characteristics to obtain protected data.

PRE primitive: In timeslot re-encryption, although a master authority must send two ciphertext components $\tilde{C}_{TS}, \tilde{C}'_{TS}$ and two update values $h^{\tilde{s}}, e(g,g)^{\alpha\tilde{s}}$ to a server, all of them are new random values used to replace or multiply (mask) the existing values, so the server cannot gain any advantages to break cryptographic functions. Note that our model is still based on the same assumption that a server can be trusted to do data operations, i.e., storing and multiplication.

5.6 Performance Evaluation and Simulations

In this section, we evaluate the cost of three off-line procedures (i.e., setup, file encapsulation, and file decausulation) and four on-line protocols (i.e., service request, token request, file access, and time slot synchronization) in terms of communication and computation cost. Then we show the performance by simulations.

5.6.1 Evaluation

Setup procedures: This procedure can be divided into two parts: First an authorizer defines underlying bilinear groups and a hash function, then computes MSK and MPK with two exponentiations on $\mathbb{G}_1$. The second part is individually performed by each owner who computes OSK and OPK with one exponentiation on $\mathbb{G}_1$.

File encapsulation: An owner performs this procedure before uploading files to cloud servers. The computation cost in this procedure results from a symmetric-key encryption for data files, a signature, and a CP-ABE encryption for the header. The first encryption depends on the size of data files, and the signature load is fixed by the signature algorithm, while the CP-ABE encryption causes $2|L| + 2$ exponentiations on $\mathbb{G}_1$, where L denotes a set of leaf nodes in an access tree.

File decapsulation: After obtaining an archive file, a consumer decrypts the header with CP-ABE, verifies a signature, and decrypts the data file. The CP-ABE decryption cost is dominated by $2|I \bigcap L| + 1$ pairing operations, where I denotes the number of

attributes in the ABE token. The verification load is also fixed by the verify algorithm. The symmetric-key decryption load also depends on the size of the data file.

Next we analyze on-line protocols in both cryptographic cost and message rounds when compared to the OAuth standard, as follows.

Service request: This protocol requires two more messages between consumers and servers, and one sign operation at a server.

Token request: The computation of CP-ABE key generation causes $2|I|+1$ exponentiations. In our scheme, this computing cost is distributed to an authorizer and an authority according to the number of confined and descriptive components. Also there are three signings (two at an authorizer, one at an authority) and three verifications (one at an owner, two at a consumer). In addition, an owner computes five pairing operations (based on our construction) to verify confined attributes she proposes. For message rounds, this protocol requires six more messages: two authorizer-authority messages, one owner-authorizer message, one consumer-owner message, and two consumer-authority messages.

File access: Unlike OAuth, our protocol has no extended message. However, for a challenge message, the cryptographic cost paid by a server is CP-ABE encryption ($2|L|+2$ exponentiations); and for a response message, the computation cost paid by a consumer is CP-ABE decryption ($2|I \bigcap L|+1$ pairing operations).

Time slot synchronization: An authorizer updates the time slot of each file until the last authorized time slot (on-demand), and a server can delay re-encryption until the file is requested (lazy re-encryption). Its computation and communication cost are four exponentiations at an authorizer, two writes and two multiplications at a server, and only one message between them. In comparison with CP-ABE encryption ($2|L|+2$ exponentiations), the cost of time slot re-encryption is much lighter.

5.6.2 Simulations

Based on the number of attributes, leaf nodes, and our construction (five confined attributes), we summarize the computation cost of on-line protocols of each entity for an ABE-token in Table 5.2.

Table 5.2 shows that the number of users and the size of the attribute universe do not affect computation cost per token. Most computation cost is expended by a consumer

Table 5.2: On-line cryptographic cost

	Signing	Verify	Exponent	Pairing
Owner		1	1	5
Consumer		2		$2(\|I \bigcap L\|) + 1$
Authorizer	2		11	
Authority	1		$2\|I - 5\|$	
Server	1		$2\|L\| + 2$	

(ASP). Some cost is expended by a cloud server, an authorizer, and an authority. Meanwhile, an owner and a authorizer pay fixed costs, and other entities' costs depend on the number of the attributes in a token or a policy.

Moreover, our protocol trades eight additional messages for stronger security: user-centric property, 2-level authentication, and end-to-end encryption/authorization. To evaluate this trade off, we compare communication cost between our scheme (AAuth) and the original standard (OAuth) without the cryptographic cost.

To this end, we have built the prototype of AAuth and OAuth protocols on the framework of OMNet++ network simulation 4.1 [92]. The simulation conditions are defined as follows: the cloud network has a bandwidth of 400 packets/second, each owner continuously requests services in exponential distribution, each service request transfers three 256 KB-files as a dummy load, the number of owners (users) ranges between 100 to 700. Figure 5.7 shows the latency time from the protocol and the dummy load.

Figure 5.7 shows the the latency time of both OAuth and AAuth and the difference of the latency time between both. The diagram shows that both the latency time and the difference increase as the number of owners increases from 100 to 700 nodes. The former observation is a general behavior of a limited-bandwidth network. The latter exhibits that the difference has no significance if the number of owners is less than 300 nodes, and the difference increases from 5 to 12 seconds after 300 nodes. The increase occurs because we limit the network bandwidth to 400 packet/seconds in our simulation model. Compared to OAuth, AAuth strikes an acceptable balance between increased network cost and improved security.

Note that we simulate OAuth and AAuth with state-full models that are closest to real implementation, so the result from simulations can represent real applications. Unfortunately, the state-full models have limitations in their number of nodes (owners). Therefore, the simulation should also be performed in stateless models that are not close to the real implementation, but the number of owners can enlarge to the size of real world situation.

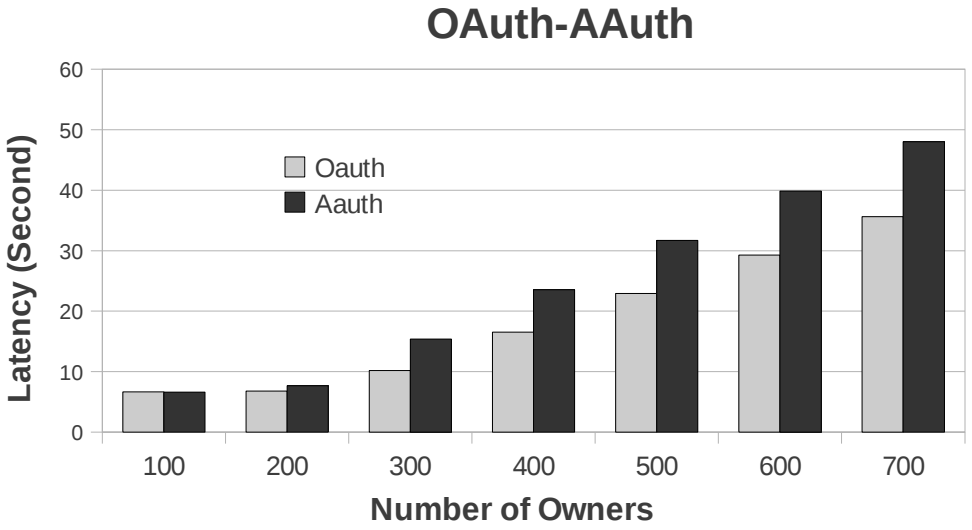

Figure 5.7: The latency time from protocol and dummy loads without cryptographic load

5.7 Summary

In this chapter, we have proposed a new authorization scheme that combats untrusted cloud servers by adopting CA-ABE, ElGamal-like masking, proxy re-encryption, and lazy re-encryption to achieve user-centric and end-to-end security. The main benefit of this scheme is that it allows users to securely share resources across providers in semi-trusted cloud environments. To achieve this end, our scheme provides 1) an ABE-token for each authorization grant, 2) a user-centric system in which an owner controls the authorization system to protect her resources, 3) end-to-end encryption and authorization from an owner to a consumer, 4) a light-weight encryption for time slot synchronization. The results of performance evaluation show that our scheme has no significant computation cost for users and is independent from the number of users in the system. Simulation results show an acceptable cost increase compensated for by better security than the current OAuth. Security analysis shows that our modified CP-ABE is as secure as the original scheme, and our protocols can resist both internal and external adversaries. Below (Table 5.3) we summarize the AAuth, showing all procedures and protocols.

Table 5.3: The procedures and protocols in AAuth

AAuth	
Procedures/Protocols	Outputs
Setup procedure	1. A bilinear group $\mathbb{G}_1, \mathbb{G}_2$; a bilinear map e; a generator g of $\mathbb{G}_1$ 2. A hash function H 3. System public & private keys MPK, MSK; Owner public & private keys OPK, OSK
File encapsulation procedure	1. An access policy $\mathbb{A}$ from both confined and descriptive attributes 2. An access tree τ 3. An archive file
Service request protocol	An access policy $\mathbb{A}$
Token request protocol	An ABE-token
File access protocol	An archive file
File decapsulation procedure	1. A header in plaintext form 2. An integrity tag 3. A data file in plaintext form
Time slot synchronization protocol	1. Two ciphertext components 2. Two update values 3. A new time slot header

Chapter 6

Conclusion and Future Research

This chapter summarizes our research contributions to the security of P2P systems and cloud computing, and gives recommendations for future research.

6.1 Summary of Contributions

The contributions of this thesis apply to two distributed systems: fully decentralized Peer-to-Peer systems and cloud computing. Although these two systems differ in their structures, they share one fundamental security vulnerability: lack of a shared trusted domain. Therefore, our contributions try to construct shared trust domains for these two environments. The first contribution is a PKI framework for P2P systems, whereas the second one is an authorization scheme for semi-trusted cloud computing.

In fully decentralized Peer-to-Peer systems, without external TTPs, no peer trusts others. In other words, each one trusts only itself. Using cryptographic tools is a common way to build security characteristics, such as confidentiality, integrity, and non-repudiation in P2P systems. However, the cryptographic tools require secret keys or public keys that are generated from trusted agencies, and fully decentralized P2P systems are naturally self-organized. Therefore, to cope with the lack of shared trust agencies in these systems, we have to create a trust framework that can work in a self-organizing fashion and is accepted by every peer in a system. Additionally, since arbitrary peers can be malicious or faulty, the trust framework must have self-healing characteristics to retain trustworthiness and resist attacks from both external and internal adversaries. Finally, this trust framework must have enough scalability to be deployed in the Internet. To achieve these goals, we

propose SOHCG, a distributed PKI framework that is self-organizing, self-healing, and scalable. SOHCG leverages CAN structure, threshold signature schemes, and the RSA homomorhic property to construct a trust group (CA group).

A CA group is a CA overlay network that automates PKI infrastructures for P2P communities but does not bundle into or depend on P2P systems. With the self-organizing structure of CAN, the defined parameters, and the group management policies, a CA group is self-organizing, dynamic, and scalable. After a bootstrapping phase, a CA group will grow up to the predefined size at the balance point (z_{max}, p_{max}), which can achieve a trade-off between efficiency and security. At the same time, membership is dynamic because of recruitment and retirement policies that enforce CA members joining and leaving the CA group. By the redundancy of CAN overloading zones, the key share of each zone is replicated by multiple peers in each zone to compensate for the lack of key tolerance in a (n, n) threshold signature scheme. Since the size of the CA group and the redundancy of key shares trade communication and computation overhead for security, a CA group copes with this potential problem by performing most protocols and coordination under a multicast group (decision group), thereby limiting communication cost and the number of participants.

To automate certificate issuing to new nodes, a CA group must form a decision group in order to verify the node ID and private-key procession of new nodes by a challenge-response scheme from uniform random locations in the overlay network. In the decision group, all decision nodes exchange their proofs with those of others, then vote on all the proof results under the threshold value $(\lfloor \frac{n-1}{3} \rfloor + \alpha n)$, which considers the effects of both external and internal adversaries. Therefore, a new node must get endorsement (i.e., certificates signed with decision nodes' shares) from all decision nodes to construct a complete certificate by leveraging the homomorphic property of the RSA signature scheme. In this way, the SOHCG can achieve its main CA functionality.

Obviously the node ID and private-key verification rely on the honest majority of a CA group. Thus, a decision group must detect its malicious members when the proof has no consensus. Because of the high communication and computation cost of the detection algorithm (a Byzantine agreement algorithm), we avoid running it frequently. To this end, we use a new node (requester) as a witness to detect whether decision nodes are suspicious before running the malicious node detection protocol.

If a decision node is malicious or a CA node's membership is expired, such a node will be eliminated from the CA group. This action can retain the honest majority and mitigate Sybil attacks. Unfortunately, this solution imposes a new security flaw because the eliminated CA nodes occupy key shares of a CA group. To strengthen the security

level of the CA group, a decision group uses the key share rendering protocol to disable the key shares known by the eliminated nodes. To achieve this, the share rendering leverages the group additive property of the (n, n) threshold scheme in each dimension of the CAN structure. Finally, to revoke the certificate of malicious nodes, SOHCG uses the same method as the certificate issuing procedure to generate CRLs and leverages P2P networks themselves as a distributed repository for storing CRLs.

Security analysis shows that SOHCG can thwart both external and internal adversaries as follows.

1. Node impersonation attacks. These attacks can be launched by an adversary issuing faked certificate requests or a coalition of adversaries launching MITM attacks. In the former case, an adversary fails to convince a CA group to accept a fake certificate request because he/she cannot receive challenge messages from the decision group. In the latter case, a coalition of adversaries fails to convince a CA group to issue a certificate if they cannot intercept more than α fraction of challenge-response messages.

2. DoS attacks. Adversaries fail to convince a decision group to run unnecessary Byzantine agreement algorithms because the decision group uses a new node as a witness to detect these attacks.

3. Sybil attacks. SOHCG leverages the *Retirement* policy and the *age* parameter to limit the membership period of CA nodes, thereby mitigating Sybil attacks.

4. CA functionality interference attacks. In these attacks, adversaries are decision nodes and try to attack two protocols: key registration, and certificate issuing. In the key registration protocol, adversaries fail to modify the certificate requests because the faked certificates can be detected by verifying the bootstrap node's signature. Adversaries fail to disrupt challenge-response voting by broadcasting invalid challenge messages in a decision group since the maximum number ($\lfloor \frac{n-1}{3} \rfloor$) of possible malicious nodes in a decision group is included in threshold value ($\lfloor \frac{n-1}{3} \rfloor + \alpha n$). In the certificate issuing protocol, if adversaries omit to issue certificates or issue incorrect certificates, SOHCG can establish a new decision group or a new node can resubmit a certificate request.

Our second contribution is in semi-trusted cloud computing. The lack of a shared trusted domain between data owners and CSPs as well as the loss of control of data are the main barriers to the adoption of clouds by businesses or individuals. For the former,

CAs or PKIs can transfer trust between owners and CSPs with public-key certificates. However, very few users (owners) have public-key certificates. Meanwhile, every user has already one or more on-line identities from ASPs or CSPs. In addition, authentication processes depend on the security level of information that users try to access. For example, academic information requires a login/password, and health information records require two-factor authentication. For the latter, cryptography technologies are typical approaches used to meet these requirements, but they impose key management problems. Currently, there are two well-known IAAA standards, i.e., OpenID and OAuth, for web application development. However neither can work in unshared trust domain environments like untrusted clouds. Thus, we cope with these problems by leveraging tokens to separate authentication layers from authorization layers and bundle access policies with ciphertext for authorization by using the modified CP-ABE. In order to retain control of data, we use ElGamel-like masks to allow data owners to contribute to token generation. As a result, we have created a distributed authorization scheme, named AAuth, to allow owners to store data in untrusted clouds and delegate authorization to consumers for accessing the owners' data. AAuth achieves the above goals by exploiting user-centric approaches, end-to-end encryption, and end-to-end authorization.

To store sensitive data in untrusted cloud storage, owners first encrypt and sign the data with symmetric keys and signature keys. Secondly, the owners encrypt the symmetric keys and verification keys by using the ABE encryption algorithm with access policies. Finally, the owners create archive files for which the headers come from the second step and the bodies come from the first step, then store the archive files in the cloud storage. Since the data is encrypted and signed by owners and the policies are bundled in ciphertext during the header encryption, untrusted cloud storage cannot subvert the confidentiality and integrity of the stored data.

When an owner requests a service from a consumer, an owner, an authorizer, and an authority must collaborate on token generation. To this end, we define two disjointed attribute sets, where a confined set is used to limit the scope and lifetime of a token, and a descriptive set is used to describe the characteristics of a consumer. The authorizer generates the confined-key components defined by the owner, and the authority itself defines and generates the descriptive-key components. Next, the owner and the authorizer collaborate to generate a common-key component. Finally, all key components, i.e., common, confined, and descriptive, are combined to form a complete token by the consumer. In this way, end users (owners) can authenticate themselves to an authorizer with any credentials based on security requirements and do not lose control of their data. In addition, descriptive attributes can certify the qualification of ASPs (consumers).

After obtaining tokens, consumers must prove that their tokens can satisfy the policies

in the required archive files. To this end, cloud storage generates challenges with the policies in the archive files then verifies the responses in order to guarantee that the consumers possess satisfactory tokens. The cloud storage will transfer archive files to the consumers only if the verification succeeds. After obtaining the archive files, the consumers use ABE keys from the ABE tokens to decrypt the headers to get symmetric keys and verification keys. Then they use the verification keys to verify the integrity of the archive files and use the symmetric keys to decrypt the ciphertext in the bodies. Thus, at the destinations (consumers), policies are enforced when the headers are decrypted, so AAuth can achieve end-to-end encryption and authorization.

Security analysis shows that AAuth can be securely performed in untrusted clouds and is able to prevent attacks both by external and internal adversaries. We first briefly analyze the situation for external attacks:

1. Eavesdropping and active attacks. Adversaries fail to launch these attacks since data is encrypted and signed in an end-to-end fashion.

2. MITM attacks. Since three key parts, i.e., common, confined, and descriptive parts, are sent to consumers through different SSL/TLS channels, adversaries have to simultaneously intercept all channels to launch MITM attacks, which is impossible in real world situations.

3. Off-line attacks. Without satisfactory tokens, adversaries cannot obtain archive files for these attacks.

Next, AAuth's participants that are semi-trusted entities are considered to be internal adversaries and analyzed as follows.

1. Cloud servers. We assume that cloud servers are curious about sensitive information they host. However, they cannot break the confidentiality or corrupt the integrity of data because the data is encrypted and signed when stored with them. Moreover, in time slot synchronization, cloud servers cannot learn any knowledge about CP-ABE keys because all components and update values that they receive are new random values with no relation to the keys.

2. The Authorizer. Without knowledge about an owner private key (OSK), the authorizer itself cannot generate tokens. In addition, the authorizer fails to fake confined keys because owners can verify the confined keys they receive.

3. Owners. Since the authorizer must authenticate owners before generating confined keys, owners fail to ask the authorizer to generate tokens in other people's names. In addition, if owners try to modify a confined key, they need to solve the discrete logarithm problems, thereby failing in the fabrication.

4. Consumers. If consumers try to modify key components associated with attributes they want, they need to solve pairing inversion problems, thereby failing in the modification.

For performance analysis, we analyze the cryptographic load of AAuth, which consists of three off-line procedures and four on-line protocols. First, the off-line procedures are analyzed as follows.

1. Setup. The authorizer computes MSK and MPK with two exponentiations, and each owner computes OSK and OPK with one exponentiation.

2. File encryption. An owner needs one symmetric encryption, one signature, and $2|L| + 2$ exponentiations for one file encryption.

3. File Decryption. A consumer requires one symmetric decryption, one verification, and $2|I \cap L| + 1$ pairing operations for one file decryption.

For on-line protocols, the cryptographic load that each AAuth participant must expend for one token is shown in Table 5.2. The results show that this load does not depend on the number of users and the size of the attribute universe. We also noticed the AAuth requires an additional eight messages, when compared to OAuth, as shown in Table 6.1. To show the effect of the additional messages in AAuth, we simulate AAuth by using OMNet++ network simulation 4.1. The simulation results in Figure 5.7 show that AAuth achieves an acceptable balance between increased network cost and improved security, when compared to OAuth.

6.2 Future Work

The security problems in P2P systems and cloud computing are a broad research area. This thesis proposes only some solutions in this area. Thus, we next present some possible enhancements and extensions of our work and give an overview of interesting issues that can be pursued in future work.

Table 6.1: The number of messages that AAuth requires more than OAuth

Protocol	Additional messages	Message flow
Service request	2	$C \rightarrow S$
Token request	2	$AZ \rightarrow AA$
	1	$O \rightarrow AZ$
	1	$C \rightarrow O$
	2	$C \rightarrow AA$
File access	—	

P2P Systems. As noted in the summary of threats to P2P system in Section 2.6, most internal attacks, such as incorrect routing updates, Sybil, eclipse, identity theft, etc., in P2P systems can be solved with certificates that bind between node IDs with public keys. Although SOHCG can build public-key certificates to cope with these internal attacks, P2P systems still have other open problems, such as free riding and data corruption, that are beyond the scope of this thesis.

1. Free riding. In P2P systems, lack of enhancing contribution methods may induce P2P communities to selfish behaviors. For example, some peers may retrieve resources from the system but not be willing to contribute to routing messages, storing content, computing jobs, or providing data/services to other peers. This situation can dramatical degrade the quality of services or disable services if a large fraction of peers engage in these misbehaviors. Currently, researchers have proposed many solutions that can be categorized into three main approaches: monetary, reciprocity, and reputation. For efficiency and reliability, these solutions require centralized agents to monitor balances and transactions, or to manage and consolidate reputation information. In addition, persistent identifiers are required for storing and managing long-term balances, maintaining the histories of peers, and preventing whitewashing attacks. The SOHCG framework can support long-term identifiers for monetary or reputation by adding supporting information, such as account information or application information in certificates. For decentralization, we have to distribute the information of balance and reputation in P2P systems themselves. The public keys from SOHCG may protect this sensitive information in hostile environments.

2. Data corruption. Adversaries may try to remove or modify information stored in systems as well as issue faked contents into systems. Without data identifiers and data authentication, adversaries can generate data with the same name as the original and try to replace the existing data by using normal P2P procedures. Adversaries

may also exploit P2P replication to deploy multiple faked versions in systems, thereby destroying data consistency. To address this concern, P2P systems must provide authentication and authorization that allow peers to verify the write permissions of requesters. To this end, the certificates from SOHCG may be used as credentials for access control in P2P systems.

Cloud computing. Data security in clouds still has many security concerns that are challenging researchers in this area. Although AAuth succeeds in authentication, authorization, data confidentiality and data integrity for semi-trusted clouds, we are interested in improving AAuth in terms of security, reliability, efficiency, and capability. In order to accelerate cloud adoption, we outline some possible extensions of AAuth and interesting issues in cloud computing as follows:

1. AAuth can control write permission by separating a header into two parts, i.e. read and write, and bundling a signing key into the write header encrypted by using a CP-ABE scheme with a write policy. However, this solution can achieve write permission control but cannot prevent cloud storage from corrupting data it hosts. To my knowledge, only data replication providing persistence and reliability can solve this problem.

2. Typically, ABE-tokens are indirectly revoked by the time slot synchronization protocol. However, if ABE-tokens are assigned with a long time slot or multiple time slots, the tokens become longterm credentials. This situation may necessitate token revocation in AAuth. There are ABE schemes that can be applied for direct key revocation, such as Attrapadung's ABE scheme [8] that is based on a conjunction between a broadcast encryption and an ABE.

3. AAuth was mainly designed for access control and inter-operation between ASPs and cloud storage. It is obvious that we can extend AAuth to support transaction-based applications like database applications by providing index or keyword searching for the meta data of records. For privacy, the index or keyword search may be implemented by cryptographic techniques, such as encryption searching or order preserving encryption.

4. In real world circumstances, it may be necessary for consumers to be certified by multiple authorities. AAuth can directly achieve this requirement with more latency time and communication overhead; therefore, we may seek for more efficient solutions. To strengthen AAuth's security and reliability, multi authorizer approaches may be implemented by multi-authority ABE schemes, such as Chase's ABE scheme [22].

APPENDICES

Appendix A

Miscellaneous Structured P2P Systems

A.1 Chord

Chord [122] uses one m-bit identifier space that is shared between a node ID and a key, and uses a DHT to partition the identifier space for each node and to map between a key and the node holding the key. The nodes form the ring topology of an overlay network by converting a node's address (IP address) to a node ID via a hash function, i.e., SHA-1, m = 160, and then the node IDs are ordered in an identifier circle modulo 2^{160} (numbering from 0 to $2^{160} - 1$). Thus, each node maintains the identifiers of its successor to connect each node into the ring. A content ID is converted to a key k via the same hash function. This key k is used to determine the node whose ID equals or follows k in the identifier space, named the **successor node** of k. The pair of the key k and its value is stored in the successor node of k. Chord can find the successor node of k in a ring topology by routing in one direction, e.g., clockwise, until it reaches the destination node. Figure A.1 shows an identifier circle with m =4 consisting of four nodes (1, 4, 11, and 12) and holding five keys (1, 2, 6, 9, and 14).

Knowing of a successor, although Chord can retrieve a $(key, value)$ pair of a content by passing around the ring via one successor to another successor until it reaches the key's successor node, this solution is inefficient because it may require traversing all nodes in the ring to reach the node holding the $(key, value)$ pair. To accelerate this process, each node x in Chord maintains m-entries of a routing table, named the **finger table**, whose entry i contain the node ID f^i that succeeds x at least 2^{i-1}, where $1 \leqslant i \leqslant m$. This node f^i is

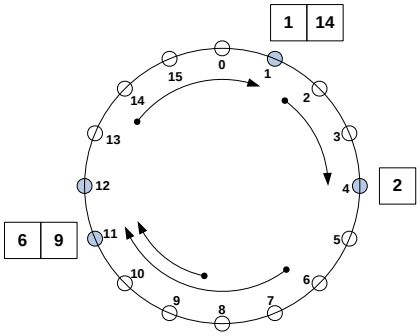

Figure A.1: An Identifier Circle of Chord with $m = 4$

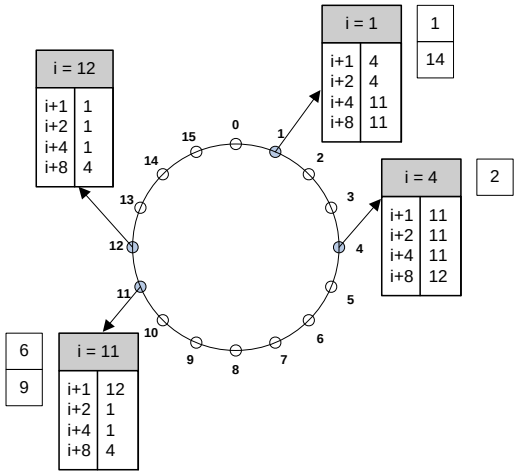

Figure A.2: The Finger Table and Key Location in the Ring Topology of Chord

called the i^{th} **finger node** of x. That is, each node stores pointers to the node at ring ways $(1, 2, 4, \ldots 2^{m-1})$ from x. Consequently, each node knows more about the nodes close to it than the nodes further away, and when a node x does not have enough information to find the node holding a key k, x asks its finger node closest to the key for a closer node and repeats this process until x can find the appropriate node. Finally, Chord can guarantee all lookup via $O(logN)$ messages, where N is the number of nodes participating in the overlay network by maintaining m entries in its finger table. Figure A.2 shows the finger table and key location in the Chord ring with nodes (1, 4, 11, and 12) and five keys (1, 2, 6, 9, and 14).

A new node x joins the Chord network by finding one exiting node and asking this node to find the successor node of node ID x. Next, the new node loads the successor list and the finger table from its successor and uses them to find the correct values for its own list and table. Finally, the new node must direct the successor nodes and other related nodes to add the new node into their finger tables. To simplify and achieve this goal, each node in Chord maintains a predecessor list, which point to the nodes that refer to it. This predecessor list is used to find the nodes that need to add the new node into their finger tables. When a node y voluntarily or involuntarily leaves Chord, nodes whose finger table include y must replace y with y's successor. To achieve this, each node must maintain r successors, i.e., a list of its next r successors. The nodes in this successor list can also provide replicated data. When a node joins or leaves the system, Chord requires $O(log^2 N)$

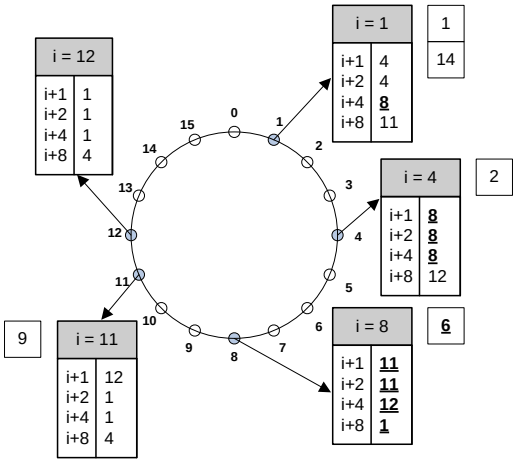

Figure A.3: The Finger Tables of Chord after Node '8' Joins

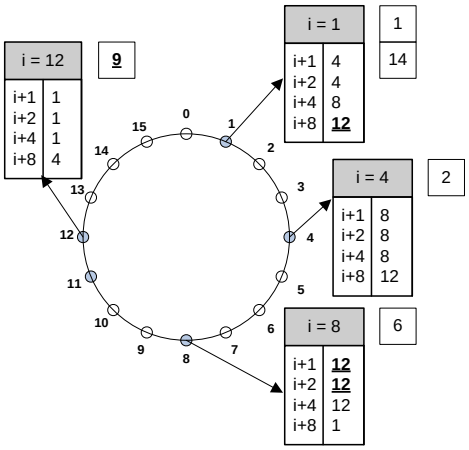

Figure A.4: The Finger Tables of Chord after Node '11' Leaves

messages to correct finger tables. The finger tables of Chord after node 6 joins and node 1 leaves are shown in Figure A.3 and Figure A.4, respectively.

A.2 Pastry

Pastry [4] uses a 128-bit identifier space for locating the position of a node ID in ring topology, whose GUID is based on a hash function that ranges from 0 to $2^{128} - 1$. When a new node joins the Pastry network, a node ID is randomly assigned to the new node with uniform distribution. Pastry routes messages to the node whose node ID numerically is closest to the given key, known as a **prefix routing**. In each routing hop, a message is forwarded to the node that shares one more digit than the current node ID. If appropriate entry in the routing table is unavailable, or available but not alive, the message is forwarded to the numerically closest node in the leaf set L of the current node. The leaf set L is a set of nodes with $|L|/2$ closest nodes numerically larger than the current node and $|L|/2$ closest nodes numerically smaller than the current node. These larger and smaller sets of the closest nodes are similar to the successor and predecessor list in the Chord network, respectively. To guarantee reaching a destination node in $O(\log_B N)$ (where B is base, N is the number of nodes participating in the overlay network), each Pastry node must maintain a routing table (similar to the finger table in Chord) consisting of d rows and $B - 1$ entries per row ($d = \log_B 128$ is the number of digit). Each entry in row n refers to a node whose

Routing Table (Node ID = 431605)							
-0xxxxx	-1xxxxx	-2xxxxx	-3xxxxx	-4xxxxx	-5xxxxx	-6xxxxx	-7xxxxx
4-0xxxx	4-1xxxx	4-2xxxx	4-3xxxx	4-4xxxx	4-5xxxx	4-6xxxx	4-7xxxx
43-0xxx	43-1xxx	43-2xxx	43-3xxx	43-4xxx	43-5xxx	43-6xxx	43-7xxx
431-0xx	431-1xx	431-2xx	431-3xx	431-4xx	431-5xx	431-6xx	431-7xx
4316-0x	4316-1x	4316-2x	4316-3x	4316-4x	4316-5x	4316-6x	4316-7x
43160-0	43160-1	43160-2	43160-3	43160-4	43160-5	43160-6	43160-7

Small	
431512	431544
431520	431563
431525	431567
431534	431600

Large	
431607	431645
431612	431650
431624	431657
431637	431702

Neighbor	
571206	417625
706354	024657
650731	157406
347621	251763

Figure A.5: The Routing Table, Neighbor Set, and Leaf Set in the Pastry Node '431605' (Notation of each Entry: common prefix - rest of ID)

ID shares the first n digits with the current node, but the node's $n + 1^{th}$ digit is different. Although the typical value of B is 16, the choice of B is a trade-off between the size of a routing table $d \times (B - 1)$ and the maximum number of hops $O(\log_B N)$. Although prefix routing can guarantee that an intermediate node is numerically closer in each step, the real path length of an underly network may be extremely different. Pastry uses a proximity metric of the underly network, such as a hop count or round trip latency measurement, to select the appropriate node when setting up the routing table. Consequently, in each routing step, a message is forwarded to a relative proximity node with a node ID that shares more numerical prefix digits with the key.

Moreover, a Pastry node maintains another set of nodes, i.e., a neighborhood set. The neighborhood set M is the set of $|M|$ nodes closest to the local node, according to the proximity metric. This set is used to maintain the locality of Pastry. To find a proximity node, Pastry includes an algorithm recursively measuring the round-trip delay by periodically sending a probe message to each member of the leaf set of the currently known node. Typically, the value of $|M|$ and $|L|$ are B or $2 \times B$. Figure A.5 shows the routing table, neighborhood set and leaf set of the Pastry node '431605' with base $B = 8$, and $d = 6$ digits.

A.3 Viceroy

Viceroy [80] is an overlay network that provides features common to other overlay networks, but also has two special ones, i.e., constant-degree and random routing. These two features facilitate three advantages: no congestion, limited join and leave cost, and short lookup path length. The constant-degree network limits the number of nodes changing their states and the load incurred in joining and leaving the system. The random routing balances load across all of the active nodes in the system. Although Viceroy employs a constant-degree network, it preserves a logarithmic path length for lookup processes. Like Chord, Viceroy maps a node object (node ID) and content object (key) on the same ring identifier space, but this ring is a unit ring [0...1) and is split into $\log N$ levels. A $(key, value)$ pair is mapped to the closest node to the key (successor node). Unlike Chord, which uses m entry in a finger table to route efficiently, Viceroy uses links between successors and predecessors on the same ring level for short distance routes and uses links between levels for long distance routes. These long distance routes are chosen randomly, with a particular bias toward closer points. From these improvements on the single-level ring of Chord, Viceroy forms an approximate butterfly network. Viceroy is formalized and proves that the routing process requires only $O(\log N)$ with nearly optimized congestion. A routing table size is $\log N$ entries. From a node joining or leaving the system, there are only $O(1)$ nodes changing their states, and the routing table is completely updated within $O(\log N)$ messages.

A.4 Kademlia

Kademlia [83] is a P2P overlay network that shares a 160-bit key space for both a node ID and a key, and a $(key, value)$ pair is stored on the node's ID closest to the key. Kademlia uses an XOR metric for the distance between points a and b in the ID space, i.e., $d(a, b) = a \bigoplus b$. The XOR metric is symmetric, i.e., $d(a, b) = d(b, a) = a \bigoplus b$, so it precisely balances load in routing and allows parallel routing to select the lowest latency path. XOR is unidirectional, i.e., for any point x and distance $d > 0$, there is an exact point y such that $d(x, y) = d$, so all lookups for the same key converge in the same direction. Each node maintains a routing table called a k bucket. This routing table consists of the *(IP Address, UDP Port, Node ID)* triples of nodes in which 2^i to 2^{i+1} is far away from itself, for each $0 < i < 160$, and these triples are sorted by time, i.e., the least recently accessed node is stored at the head, and the most recently accessed node is stored at the tail. Kademlia uses PING, STORE, FIND_NODE, and FIND_VALUE messages for its routing. To locate the k closest nodes to a given node ID, node a ping n nodes from its closest k-bucket, then

parallel sends FIND_NODEs to the n nodes to find a node closer than the n nodes. To get a $(key, value)$ pair, a node sends FIND_VALUEs to n nodes whose ID is closest to the key.

A.5 Tapestry

Tapestry [136] provides Decentralized Object Location and Routing (DOLR), which focus on routing to mobile or replicated objects in the presence of instability in the underlying network. In order to achieve this goal, objects are named by identifiers that are encoded without knowledge of physical locations, and message routing uses only localized knowledge. While other P2P systems, such as, CAN and Chord, fix the number and location of contents and replicas by a DHT, Tapestry allows an application to place location pointers of a replica throughout the network to facilitate efficient routing to the associated content. Tapestry's identifier space is 160 bits with a hexadecimal digit (40-digit hexadecimal identifier) by using SHA-1. Similar to Pastry, both node and content share the same Globally Unique Identifiers (GUID) and the prefix routing algorithm is used for routing. To reach the destination in $O(log_b N)$, Tapestry routes to destination ID by forwarding to the next node whose ID is progressively closer to the destination ID digit by digit, i.e., $4 * ** \Rightarrow 42 * * \Rightarrow 42A* \Rightarrow 42AD$, where * denotes a wild-card.

To publish content in Tapestry, a server s, which stores content with GUID o, periodically publishes its content by routing a publish message to the root node o_r of content o. Each node along the publication path stores a location pointer (o, s). If there are multiple servers storing the same content o, each server must publish individually. The nodes along the overlap publishing path of multiple servers maintain multiple location pointers for replication. Figure A.6 shows the process by which two copies of content '450383' are published to their root node '45038-1' by two servers '45-4126' and '654218'.

Each node queries content o by routing a lookup message to root node o_r of the content. Each node along the querying path looks for its local mapping point for content o. If the mapping point of o is found, the lookup message will be redirected to the server directly. Otherwise, the lookup message will be forwarded until it reaches the root node of o. With this method, the requesting nodes closer to the path will cross the publishing path sooner, and they will reach the content faster. Consequently, Tapestry can look up content more efficiently. Figure A.7 shows several nodes sending lookup messages for content '450383'.

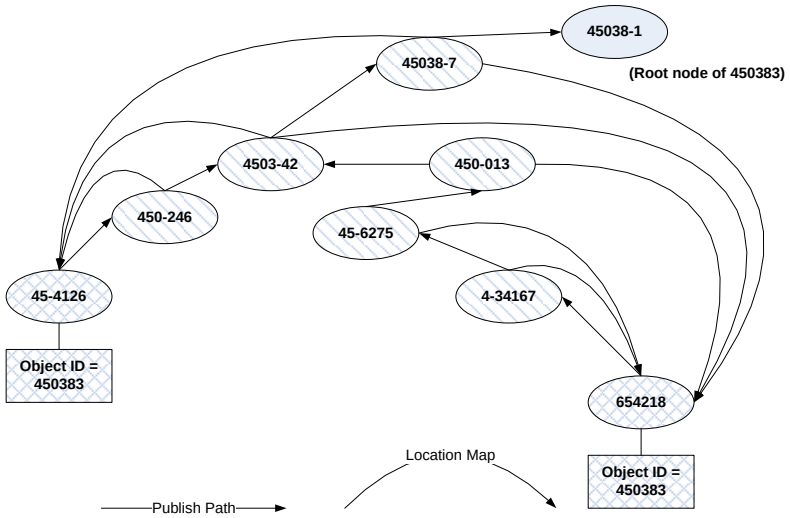

Figure A.6: The Publish Paths and Location Pointers of each Sever in Tapestry

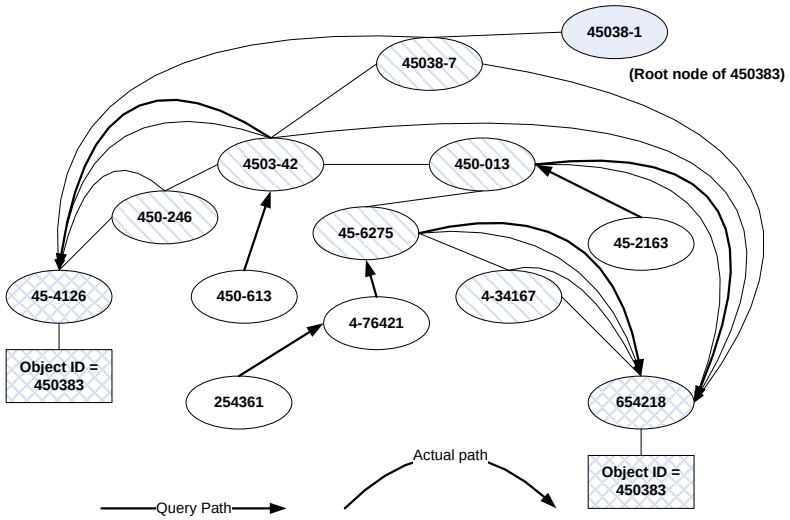

Figure A.7: The Query Process and the Actual Path in Tapestry

Bibliography

[1] "Active Directory Server Identity Credential Protection", *http://www.microsoft.com/en-us/server-cloud/windows-server/active-directory.aspx*.

[2] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, R. P. Wattenhofer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment", In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI '02)*, 2002.

[3] Amazon Web Services, *http://aws.amazon.com*.

[4] I. Antony, T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems", *Middleware '01*, R.Guerraoui(ED.), LNCS 2219, pp. 329-350, Springer, 2001.

[5] AIM-Chat with all your buddies & Facebook friends, *http://www.aim.com*.

[6] Google cloud services-App Engine,*http://www.google.com/enterprise/cloud/appengine*.

[7] N. Arora and R.K. Shyamasundar, "PGSP: a protocol for secure communication in peer-to-peer system" *Wireless Communications and Networking Conference, IEEE*, vol. 4, pp. 2094-2099 ,2005.

[8] N. Attrapadung and H. Imai: Conjunctive Broadcast and Attribute-Based Encryption, In *Conference on Paring-Based Cryptography*, vol. 5671 of LNCS, pp. 248-265, Springer, 2009.

[9] A. Avramidis, P. Kotzanikolaou and C. Douligeris,"Chord-PKI: Embedding a Public Key Infrastructure into the Chord Overlay Network", *EuroPKI 2007: Public Key Infrastructure, Springer-Verlag*, J. Lopez, P. Samarati, and J.L. Ferrer (Eds.), LNCS 4582, pp. 354-361, 2007.

[10] Windows Azure Platform, *http://www.microsoft.com/windowsazure*.

[11] A. Beimel, "Secure Schemes for Secret Sharing and Key Distribution", *PhD thesis, Israel Institute of Technology*, Technion, Haifa, Israel, 1996.

[12] J. C. Benaloh, "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret." *Advances in Cryptology CRYPTO 86, Springer-Verlag*, A.M. Odlyzko (Eds.), LNCS 263, pp. 251-260, 1987.

[13] M. Ben-Or, s. Goldwasser, A. Wigderson, "Completeness theorems for non-cryptographic fault tolerant distributed computation", In Proceeding of 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 1-10, 1988.

[14] J. Bethencourt, A. Sahai and B. Waters, "Ciphertext-Policy Attribute-Based Encryption", In *IEEE Symposium on Security and Privacy*, pp. 321-334, 2007.

[15] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocol and atomic proxy cryptography", In *Proceedings of EUROCRYPT'98*, vol. 1402, pp. 127-144, 1998.

[16] D. Boneh and M. Franklin, "Efficient generation of shared RSA keys", *J.ACM, New York, NY, USA*, Vol. 48, Iss. 4, pp. 702-722, 2001.

[17] K. D. Bowers, A. Juels and A. Opera, "Proof of Retrievability: Theory and Implementation", In *ACM CCSW 2009*, 2009.

[18] K. D. Bowers, A. Juels and A. Opera, "HAIL: A High-Availability and Integrity Layer for cloud Storage", In *ACM CCS 2009*, 2009.

[19] Box.net-online file sharing, content management, collaboration, *http://www.box.net*.

[20] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, Springer-Verlag, Berlin, Heidelberg, 2003.

[21] M. Castro, P. Druschel, A. Ganesh , A. Rowstron and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks" In *Proceeding of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, UNIX, 2002.

[22] M. Chase, "Multi-authority Attribute Base Encryption", In *TCC 2007*, vol. 4392 of LNCS, pp. 515-534, Springer, 2007.

[23] R. Chen, W. Guo, L. Tang, J. Hu and Z. Chen, "Scalable Byzantine Fault Tolerant Public Key Authentication for Peer-to-Peer Networks", *Euro-Par 2008: Parallel Processing, Springer-Verlag*, E. Luque, T. Margalef, and D. Bentez (Eds.), LNCS 5168, pp. 601-610, 2008.

[24] L. L. Chen and G. Gong, *Course Note in ECE 493 Topic 10: Communication System Security*, University of Waterloo, CA, 2009.

[25] I. Clarke, O. Sandberg, B. Wiley and T. W. Hong, "Freenet: a distributed anonymous information storage and retrieval system", *Anonymity 2000: International workshop on Designing privacy enhancing technologies, Springer-Verlag*, H. Federrath (Ed.), LNCS 2009 ,pp. 46-66, 2001.

[26] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System" *Designing Privacy Enhancing Technologies*, LNCS 2009, PP. 46-66, 2001.

[27] Cloud Computing Interoperability Forum (CCIF), *http://cloudforum.org*.

[28] CloudStandards, *http://cloud-standards.org*.

[29] F. Cristain and C. Fetzer, "The timed asynchronous distributed system model", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 6, pp. 642-657, 1999.

[30] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, "Wide-area cooperative storage with CFS", In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01)*, 2001.

[31] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*, Springer-Verlag, Berlin, Heidelberg, 2007.

[32] H. Deng, A. Mukherjee and D. P. Agrawal, "Threshold and identity-based key management and authentication for wireless ad hoc networks", *ITCC 2004: Proceedings of International Conference on Information Technology: Coding and Computing*, vol. 1, pp. 107-111, 2004.

[33] Y. Desmedt and Y. Frankel, "Threshold cryptosystems", *CRYPTO 89: Advances in Cryptology, Springer-Verlag*, G. Brassard (Ed.), LNCS 435, pp. 307-315, 1990.

[34] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures", *CRYPTO 91: Advances in Cryptology, Springer-Verlag*, J. Feigenbaum (Ed.), LNCS 576, P. 457-469, 1992.

[35] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)", *Internet Engineering Task Force (IETF)*, August 2008.

[36] Cloud—DMTF, *http://dmtf.org/standards/cloud*.

[37] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement", *SIAM Journal of Computing*, vol. 12, no. 4, pp. 656-666, 1983.

[38] J. R. Douceur, "The Sybil Attack", *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag*, P. Druschel, F. Kaashoek, and A. Rowstron (Eds.), LNCS 2429, pp. 251-260, 2002.

[39] Z. Duan, K. Gopalan and Y. Dong, "Push vs. pull: implications of protocol design on controlling unwanted traffic", In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05)*, USENIX, 2005.

[40] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *Transactions on Information Theory, IEEE*, vol. 31, no. 4, pp. 469-472, 1985 J. Feigenbaum (Ed.), LNCS 576, pp. 457-469, 1992.

[41] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing", *28th Annual Symposium on Foundations of Computer Science, IEEE*, pp. 427-437, 1987.

[42] Y. Frankel, P. Gemmell, P. D. MacKenzie and M. Yung, "Optimal-resilience proactive public-key cryptosystems", *Proceedings of 38th Annual Symposium on Foundations of Computer Science, IEEE*, pp. 384-393, 1997.

[43] L. Ganesh and B. Y. Zhao, "Identity theft protection in structured overlays" *1st IEEE ICNP Workshop on Secure Network Protocols (NPSec)*, pp. 49-54, 2005.

[44] H. Garcia-Molina, "Elections in distributed computer systems", *IEEE Transactions on Computers*, Vol. C-31, No. 1, pp. 48-49, 1982.

[45] C. George, D. Jean and K. Tim, *Distribute Systems: Concepts and Design*, Pearson Education Limited, 2005.

[46] Cloud Infrastructure from GoGrid, *http://www.gogrid.com*.

[47] Google Apps for Business, *http://www.google.com/apps*.

[48] Google Docs, *http://www.google.com/google-d-s/documents*.

[49] V. Goyal, O. Pandey, A. Sahai and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data", In *ACM conference on Computer and Communications Security (ACM CCS)*, pp. 89-98, 2006.

[50] B. Grobauer, T. Walloschek, E. Stocker, "Understanding Cloud Computing Vulnerabilities" *IEEE Security and Privacy*, Vol. 9, Issue 2, pp. 50-57, 2010.

[51] Apache Hadoop, *http://hadoop.apache.org*.

[52] , S. B. Handurukande, A. M. Kermarrec, F. L. Fessant, L. Massoulie and S. Patarin, "Peer Sharing Behaviour in the eDonkey Network, and Implications for the Design of Server-less File Sharing Systems", In *Proceedings of ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '06)*, 2006.

[53] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems", *ITCC 2005: International Conference on Information Technology: Coding and Computing*, Vol. 2, pp. 205-213, 2005.

[54] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk and M. Yung, "Proactive public key and signature systems", *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security, ACM*, pp. 100-110, 1997.

[55] C.N. Hoefer and G. Karagiannis, "Toxonomy of cloud computing services" *IEEE Globecomm 2010 Workshop on Enabling the Future Service-Oriented Internet*, 2010.

[56] R. Housley, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", *http://www.ietf.org/rfc/rfc2459.txt*, RFC 2459, 1999.

[57] IBM Cloud Computing, *http://www.ibm.com/cloud-computing*.

[58] ICQ 7.6, *http://www.icq.com*.

[59] Intel Virtualization Technology (Intel VT), *http://www.intel.com/technology/virtualization/technology.htm*.

[60] ISO/IEC 27005:2007 "Information Technology–Security Techniques–Information Security Risk Management", Int'1 Org. Standardization, 2007.

[61] Jabber.org, *http://www.jabber.org*.

[62] M. Jelasity, A. Montresor, G. P. Jesi and S. Voulgaris,"PeerSim: A Peer-to-Peer Simulator",*http://peersim.sourceforge.net*, 2006.

[63] M. Jensen, J. Schwenk, N. Gruschka and L. L. Iacono, "On Technical Security Issues in Cloud Computing", *IEEE International Conference on Cloud Computing*, 2009.

[64] L. Jin, "On peer-to-peer (P2P) content delivery", *Peer-to-Peer Networking and Applications, Springer US*, X. Shen and H. Yu (Eds.), Vol. 1, Iss. 1, pp. 45-63, 2008.

[65] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus-scalable secure file sharing on untrusted storage", In *Proceedings of Second USENIX Conference on File and Storage Technologies*, March 2003.

[66] C. Kalt, "Internet Relay Chat: Client Protocol (RFC 2812)", *Internet Engineering Task Force (IETF)*, April 2000.

[67] C. Kalt, "Internet Relay Chat: Server Protocol (RFC 2813)", *Internet Engineering Task Force (IETF)*, April 2000.

[68] C. Neuman, S. Hartman and K. Raeburn, "The Kerberos Network Authentication Service (V5)" (RFC 4120), *Internet Engineering Task Force (IETF)*, July 2005.

[69] J. Kong, P. Zerfos, H. Luo, S. Lu. and L. Zhang, "Providing robust and ubiquitous security support for mobile ad-hoc networks", *Proceeding of Ninth International Conference on Network Protocols, IEEE*, pp. 251-260, 2001.

[70] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, B. Zhao, "OceanStore: an architecture for global-scale persistent storage", In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems (ASPLOS-IX)*, 2000.

[71] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Language and Systems*, vol. 4, no. 3, pp. 382-401, 1982.

[72] N. Leibowitz, M. Ripeanu and A. Wierzbicki, "Deconstructing the Kazaa network", *WIAPP 2003: Proceedings of the Third IEEE Workshop on Internet Applications*, pp. 112-120, 2003.

[73] F. Lesueur, L. Ḿe and V. V. T. Tong, "A Distributed Certification System for Structured P2P Networks", *IFIP International Federation for Information Processing 2008, Resilient Networks and Services, Springer-Verlag*, D. Hausheer and J. Schonwalder (Eds.), LNCS 5127, pp. 40-52, 2008.

[74] F. Lesueur, L. Ḿe and V. V. T. Tong, "An efficient distributed PKI for structured P2P networks", *P2P'09: IEEE Ninth International Conference on Peer-to-Peer Computing 2009*, pp. 1-10 , 2009.

[75] , J. Liang, R. Kumar, K. W. Ross, "The FastTrack overlay: A measurement study" *Computer Networks*, Vol. 50, Iss. 6, PP. 842-858, 2006.

[76] Y. Liu, Y. Guo and C. Liang, "A survey on peer-to-peer video streaming systems", *Peer-to-Peer Networking and Applications, Springer US*, X. Shen and H. Yu (Eds.), Vol. 1, Iss. 1, pp. 18-28, 2008.

[77] H. Liu, "A New Form of DOS Attack in a Cloud and Its Avoidance Mechanism", *Cloud Computing Security Workshop (CCSW)*, 2010.

[78] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes", *Communications Surveys and Tutorials, IEEE*, vol. 7, no. 2, pp. 72-93, 2005.

[79] Y. Mao, V. Narayanan, A. Swaminathan, "Threat Analysis for Peer-to-Peer Overlay Networks(draft-mao-p2psip-threat-analysis-00), *Internet Engineering Task Force (IETF)*, March 2009.

[80] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly", *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing, ACM*, pp. 183-192, 2002.

[81] Google Research Publication:MapReduce, *http://labs.google.com/papers/mapreduce.html*.

[82] R. Matei, A. Iamnitchi and P. Foster, "Mapping the Gnutella network", *Internet Computing, IEEE*, vol. 6, no. 1, pp. 50-57, 2002.

[83] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag*, P. Druschel, F. Kaashoek, and A. Rowstron (Eds.), LNCS 2429, pp. 5365, 2002.

[84] P. Mell and T. Grance, "The NIST Definition of Cloud Computing (Version 15)" National Institute of Standards and Technology, Information Technology Laboratory, 2009.

[85] Window Live, *http://explore.live.com*.

[86] Napster Page, *http://free.napster.com*.

[87] K. V. Nguyen, "Simplifying Peer-to-Peer Device Authentication Using Identity-Based Cryptography" *ICNS '06: Proceedings of the International conference on Networking and Services, IEEE Computer Society*, pp. 43-47, 2006.

[88] M. Nystrom and B. Kaliski, "PKCS 10: Certification Request Syntax Specification Version 1.7", *http://www.ietf.org/rfc/rfc2986.txt*,RFC 2986, 2000.

[89] E. Hammer-Lahav, "The OAuth 1.0 Protocol (RFC 5849)", *Internet Engineering Task Force (IETF)*, April 2010.

[90] E. Hammer-Lahav, D. Recordon, D. Hardt, "The OAuth 2.0 Authorization Protocol" (Draft-ietf-oauth-v2-15), *Internet Engineering Task Force (IETF)*, April 2011.

[91] Open Cloud Consortium (OCC), *http://opencloudconsortium.org*.

[92] OMNet++ Network Simulation Framework, *http://www.omnetpp.org*.

[93] specs@openid.net, "The OpenID Authentication 2.0-Final" *http://openid.net/specs/openid-authentication-2_0.html*, December 2007.

[94] Oracle Public Cloud, *http://cloud.oracle.com*.

[95] R. Ostrovsky, A. Sahai, B. Waters, "Attribute-Based Encryption with Non-Monotonic Access Structure", In *ACM conference on Computer and Communications Security (ACM CCS)*, pp. 195-203, 2007.

[96] V. Pathak and L. Iftode, "Byzantine fault tolerant public key authentication in peer-to-peer systems", *Computer Networks*, vol. 50, no. 4, pp. 579-596, 2006.

[97] Pig Yahoo Research, *http://research.yahoo.com/node/90*.

[98] D. C. Plummer, T. J. Bittman, T. Austin, D. W. Cearley and D. M. Smith, "Cloud Computing: Defining and Describing an Emerging Phenomenon", *Gartner, Inc.*, ID Number: G00156220, June 2008.

[99] Python Programming Language, *http://www.python.org*.

[100] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", In P*roceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04)*, Vol. 34 Iss. 4, 2004.

[101] M. Rabin, "Randomized byzantine generals", *24th Annual Symposium on Foundations of Computer Science, IEEE*, pp. 403-409, 1983.

[102] Cloud Computing Cloud Hosting & online storage by Rackspace, *http://www.rackspace.com/cloud*.

[103] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker, "A scalable content-addressable network", *SIGCOMM'01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM*, pp. 161-172, 2001.

[104] S. Ratnasamy, M. Handley, R. Karp and S. Shenker' "Application-Level Multicast Using Content-Addressable Networks", *NGC 2001: Networked Group Communication, Springer-Verlag*, J. Crowcroft and M. Hofmann (Eds.), LNCS 2233, pp. 14-29, 2001.

[105] R. Ravi and S. K. Sandeep, "P2P-IM: A P2P Presence System for the Internet", *P2P 2007: Seventh IEEE International Conference on Peer-to-Peer Computing*, pp. 233-234, 2007.

[106] B.P. Rimal, E. Choi"Toxonomy and Survey of Cloud Computing Systems", *Fifth International Joint Conference on INC, IMS and IDC*, pp. 44-51, 2009.

[107] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network", In *Proceedings of The First International Conference on Peer-to-Peer Computing*, pp.99-100, Aug 2001.

[108] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications, ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[109] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01)*, 2001.

[110] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption", In *Advance in Cryptology—EUROCRYPT*, vol. 3494 of LCNS, pp. 457-473, Springer, 2005.

[111] CRM, Manage sales-Salesforce.com, *https://www.salesforce.com*.

[112] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol (RFC4511)", *The Internet Engineering Task Force (IETF)*, 1997.

[113] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol (RFC 4511)" *Internet Engineering Task Force (IETF)*, 2006.

[114] A. Shamir, "How to share a secret", *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[115] A. Singh, M. Castro, P. Druschel and A. Rowstron, "Defending against Eclipse attacks on overlay networks", In *Proceedings of SIGOPS European Workshop*, Belgium, 2004.

[116] K. Singh, H. Schulzrinne, Peer-to-peer internet telephony using SIP,*NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pp. 63-68, 2005.

[117] A. Singh, T-W. Ngan, P. Druschel and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses", *INFOCOM 2006: Proceedings of 25th IEEE International Conference on Computer Communications*, pp. 1-12, 2006.

[118] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables", *IPTPS 2002: In Proceeding of 1st International Workshop on Peer-to-Peer Systems, Springer-Verlag*, P. Druschel, F. Kaashoek, and A. Rowstron (Eds.), LNCS 2429, pp. 261-269, 2002.

[119] IBM Smart Cloud, *http://www.ibm.com/cloud-computing*.

[120] Photo Sharing-SmugMug, *http://www.smugmug.com*.

[121] J. G. Steiner, B. C. Neuman and J. I. Schiller, "Kerberos: An authentication service for open network systems", *In Proceedings of the Winter 1988 Usenix Conference*, pp. 191-201, 1988.

[122] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, ACM*, pp. 149-160, 2001.

[123] SuccessFactors Business Executive Software, *http://www.successfactors.com*.

[124] K. Takaragi, K. Miyazaki and M. Takahashi, "IEEE P1363: A Threshold Digital Signature Issuing Scheme without Secret Communication", *http://grouper.ieee.org/groups/1363/StudyGroup/contributions/th-sche.pdf*, IEEE P1363, 1998.

[125] A. Takeda, K. Hashimoto, G. Kitagata, S.M.S. Zabir, T. Kinoshita and N. Shiratori' "A New Authentication Method with Distributed Hash Table for P2P Network", *AINAW 2008: 22nd International Conference on Advanced Information Networking and Applications - Workshops*, pp. 483-488, 2008.

[126] G. M. Thomer, K. Frans, L. Jinyang, M. Robert and S. Jeremy, "P2PSim: A simulator for peer-to-peer protocol", *http://pdos.csail.mit.edu/p2psim*, 2005.

[127] B. Waters, "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization", In *PKC: Public Key Cryptography*, vol 6571 of LNCS, pp. 53-70, Springer, 2011.

[128] J. Wei, X. Zhang, G. Ammons, V. Bala and P. Ning, "Managing Security of Virtual Machine Images in a Cloud Environment", *Cloud Computing Security Workshop (CCSW)*, 2009.

[129] Xen Hypervisor, *http://xen.org*.

[130] Yahoo, *http://www.yahoo.com*.

[131] Yahoo! Messenger-Chat,Instant message, SMS, Video call, PC calls, *http://messenger.yahoo.com*.

[132] L. Youseff, M. Butrico, D. Da Silva, "Toward a Unified Ontology of Cloud Computing" *Grid Computing Environments Workshop (GCE '08)*, 2008.

[133] S. Yu, C. Wang, K. Ren, and W. Lou, "Acieveing Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing", In *Proceeceding of IEEE INFOCOM'10*, pp. 534-542, 2010.

[134] A. Yun, C. Shi and Y. Kim, "On Protecting Integrity and Confidentiality of Cryptographic File System for Outsource Storage", In *ACM CCSW 2009*, 2009.

[135] S. Zarandioon, D. Yao, and V. Ganapathy, "K2C: Crytographic Cloud Storage with Lazy Revocation and Anonymous Access", In *International ICST Conference on Security and Privacy in Communication Networks (Securecomm'11)*, 2011.

[136] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment", *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41-53, 2004.

[137] P. R. Zimmermann, *The official PGP user's guide*, *MIT Press, Cambridge, MA, USA,*, 1995.

[138] W. Wang, Z. Li, R. Ownes and B. Bhargava, "Secure and Efficient Access to Outsource Data", In *ACM CCSW 2009*, 2009.