Highly Scalable and Secure Mobile Applications in Cloud Computing Systems

by

Piotr Konrad Tysowski

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, 2013

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

Cloud computing provides scalable processing and storage resources that are hosted on a third-party provider to permit clients to economically meet real-time service demands. The confidentiality of client data outsourced to the cloud is a paramount concern since the provider cannot necessarily be trusted with read access to voluminous sensitive client data. A particular challenge of mobile cloud computing is that a cloud application may be accessed by a very large and dynamically changing population of mobile devices requiring access control. The thesis addresses the problems of achieving efficient and highly scalable key management for resource-constrained users of an untrusted cloud, and also of preserving the privacy of users. Computation and wireless communication is minimized for mobile users while preserving the confidentiality of cloud data and of users retrieving it.

A model for key distribution is first proposed that is based on dynamic proxy re-encryption of data. Keys are managed inside the client domain for trust reasons, computationally-intensive re-encryption is performed by the cloud provider, and key distribution is minimized to conserve communication. A mechanism manages key evolution for a continuously changing user population.

Next, a novel form of attribute-based encryption is proposed that authorizes users based on the satisfaction of required attributes. The greater computational load from cryptographic operations is performed by the cloud provider and a trusted manager rather than the mobile data owner. Furthermore, data re-encryption may be optionally performed by the cloud provider to reduce the expense of user revocation.

Another key management scheme based on threshold cryptography is proposed where encrypted key shares are stored in the cloud, taking advantage of the scalability of storage in the cloud. The key share material erodes over time to allow user revocation to occur efficiently without additional coordination by the data owner; multiple classes of user privileges are also supported.

Lastly, an alternative exists where cloud data is considered public knowledge, but the specific information queried by a user must be kept private. A technique is presented utilizing private information retrieval, where the query is performed in a computationally efficient manner without requiring a trusted third-party component. A cloaking mechanism increases the privacy of a mobile user while maintaining constant traffic cost.

All proposed algorithms and protocols have been implemented on popular commercial mobile and cloud computing platforms to demonstrate feasibility and provide real-world performance benchmarks. The scalability potential of the various schemes is also shown through simulations. Options are presented throughout for adapting the techniques to the unique requirements of various types of cloud systems.

Acknowledgements

My research projects would not have been possible without a number of people to whom I wish to express my sincere and utmost gratitude.

I thank my Ph.D. supervisor, Professor Anwar Hasan, of the Electrical and Computer Engineering Department in the University of Waterloo, for his invaluable advice, support, and guidance throughout my four years at the university for which I am greatly indebted.

I am also grateful to the members of my Ph.D. examination committee, including Professors Sagar Naik and Paul Ward of the Electrical and Computer Engineering Department, and Professor Urs Hengartner of the David R. Cheriton School of Computer Science, in the University of Waterloo, for their indispensable feedback on my proposal and thesis, and collaborations on research work.

I am very appreciative of my external examiner, Professor Kui Ren, of the Department of Computer Science and Engineering, in the State University of New York at Buffalo, for kindly traveling to Waterloo to participate as a committee member in my Ph.D. defence.

I also thank my additional publication co-authors Professor Ian Goldberg of the David R. Cheriton School of Computer Science, and fellow graduate students Femi Olumofin and Tony Zhao in the University of Waterloo, for their collaborations on the work related to my thesis, and Ayush Gupta, for his assistance with programming work on an earlier research topic as an undergraduate research assistant.

I also extend thanks to my family for their endless support and understanding.

My research work was supported in part by a National Sciences and Engineering Research Council (NSERC) grant awarded to Professor Anwar Hasan, and an NSERC Alexander Graham Bell Canada Graduate Scholarship, a University of Waterloo President's Graduate Scholarship, and University of Waterloo Graduate Scholarships awarded to me.

This thesis is dedicated to my mother and father.

Table of Contents

List of Tables

List of Figures

Nomenclature

3G	Third Generation
4G	Fourth Generation
ABE	Attribute-Based Encryption
ACL	Access Control List
AES	Advanced Encryption Standard
BDH	Bilinear Diffie-Hellman
CA	Central Authority
CBC	Cipher Block Chaining
CLPKC	Certificateless Public Key Cryptography
CP-ABE	Ciphertext-Policy Attribute-Based Encryption
CPU	Central Processing Unit
CSP	Cloud Service Providers
DHT	Distributed Hash Table
DMS	Degrees-Minutes-Seconds
EC2	Elastic Compute Cloud
ECDSA	Elliptic Curve Digital Signature Algorithm

EDGE	Enhanced Data for Global Evolution
GAE	Google App Engine
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HIBE	Hierarchical Identity-Based Encryption
HSDPA	High-Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IBE	Identity-Based Encryption
IEEE	Institute of Electrical and Electronics Engineers
ISP	Internet Service Provider
IT	Information Technology
J2SE	Java 2 Standard Edition
JCE	Java Cryptography Extension
jPBC	Java Pairing-Based Cryptography
JSON	JavaScript Object Notation
KGC	Key Generation Centre
KMIP	Key Management Interoperability Protocol
KP-ABE	Key-Policy Attribute-Based Encryption
LAN	Local Area Network
LBS	Location Based Service
LRWPAN	Low-Rate Wireless Personal Area Network

MCC	Mobile Cloud Computing
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operating System
PaaS	Platform as a Service
PBC	Pairing-Based Cryptography
PIR	Private Information Retrieval
PKCS	Public-Key Cryptography Standard
PKE	Public Key Encryption
PKG	Private Key Generator
PKI	Public Key Infrastructure
POI	Point Of Interest
RAID	Redundant Array of Independent Disks
RAM	Random-Access Memory
RF	Radio Frequency
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SAP	SSL Authentication Protocol
SC	Secure Coprocessor
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SHA	Secure Hash Algorithm

SMS	Short Message Service
SPIR	Symmetric Private Information Retrieval
SSL	Secure Sockets Layer
SSO	Single Sign-On
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
VHC	Various-Size-Grid Hilbert Curve
VPN	Virtual Private Network
XaaS	Everything as a Service

Chapter 1

Introduction

1.1 Cloud Computing Services

Cloud computing has garnered much interest in recent years in the computing industry, the media, and academia. It is a form of pay-per-use distributed computing consisting of data centres providing commodity resources for massively scalable units of computing and storage for commercial enterprise applications as well as scientific computing; these facilities are delivered as a service to a global population of users over the Internet and wireless data networks.

Cloud computing promises many benefits to the IT profession: the ability to scale resources to meet varying customer demand in real-time, to deliver new computing services faster, and to significantly lower capital and operational costs. Because the computing resources of a cloud are operated by a third-party, clients are relieved from the burdens of hardware ownership, maintenance, and administration of the underlying services. Clients are only responsible for deploying the applications executed in the cloud and paying for the actual consumption of network and computing resources; they need not incur the capital expenditure of hardware with excess capacity to guarantee performance during peak demand. Additionally, they need not incur the costs of maintenance, data backup, and security. Suitable applications for cloud computing include financial market modelling, scientific applications, speech recognition and synthesis, and social networks.

Due to the highly accessible nature of cloud servers, the ever-increasing capabilities

of mobile device hardware, and the availability of faster wireless networks, many cloud applications today are accessed by users of mobile devices such as smartphones. The evolution into *Mobile Cloud Computing* (MCC) has further broadened the usefulness of cloud applications, which can deliver services at any time and to any location.

In this chapter, various facets of cloud computing are introduced. In this section, cloud services are classified and their commercial advantages are summarized. Security issues that stem from cloud feature characteristics are stated in Section 1.2. An overview of contributions made in this thesis appears in Section 1.3. Finally, an outline of the following chapters of the thesis is given in Section 1.4.

1.1.1 Functional Classification of Cloud Services

Cloud Service Providers (CSPs) offer numerous services to clients [69] [68] [70]; the taxonomy may be referred to as *Everything as a Service (XaaS)*.

In *Infrastructure as a Service (IaaS)*, computational resources such as data storage and processors are made available on demand. A task may be replicated and distributed across many processors to accelerate computation. Clients gain access to virtual servers on which to deploy their own software, and the infrastructure is delivered over the Internet. The client may have control over the choice of OS, storage, and host firewalls.

In *Platform as a Service (PaaS)*, a cloud provider offers a platform that enables rapid development and deployment of scalable applications without the need for investment in an infrastructure. Higher-level services are offered, sometimes in specific domains; these include application frameworks, developer ecosystems, collaboration tools, and storefronts.

In *Software as a Service (SaaS)*, complete end-user applications are deployed, managed, and delivered over the Internet in turnkey fashion; this is made possible by dynamic web applications and standards-compliant browsers that do not require thick clients. As a result, development costs and deployment lag are reduced. Examples include customer relationship management middleware, data warehousing, and collaboration tools.

1.1.2 Advantages of Cloud Computing

It is useful to consider the economic drivers behind cloud computing, as they are largely responsible for its technical evolution and likely future paths of development. The benefits of cloud computing are numerous, and are summarized below [24] [68]:

- *Greater cost efficiencies.* A client organization does not need to acquire, provision, and manage its own computing resources; instead, it can rent the use of these resources from a cloud provider. Worldwide IT cloud services continue to grow at several times the rate of traditional IT offerings [42]. Small firms in particular can take advantage of *utility pricing*, allowing them to meet real-time needs of computation and storage. The provider undertakes installation, maintenance, and upgrades, and has expertise in the area; so, there is no need for the client to incur these expenses. Another advantage of outsourcing computation to clouds is the achievement of *green IT*, where the total resource consumption footprint is reduced.

- *Rapid deployment.* Clients are not encumbered by the deployment lag inherent in an in-house solution. The client typically provisions applications directly using a management console. Additionally, a number of cloud vendors offer a platform for the development of distributed applications, and storage solutions that scale well; for instance, Google has pioneered BigTable, a distributed storage system for managing structured data that can scale to petabytes of data across thousands of commodity servers [23]. Cloud providers offer automation tools and scripting systems that simplify the programming effort, and use standard web service protocols.

- *On-demand scalability.* A client of a cloud provider can address changes in demand for its processing by replicating applications to many concurrent runtime instances. Unanticipated burst demands such as flash traffic on a web server may be met automatically without noticeable delay. Cloud computing is particularly suitable for non-uniform workloads, where the client would otherwise acquire excess infrastructure to account for peak usage to avoid outages; this practice can lead to idle cycles.

- *Easy connectivity.* Cloud applications may be accessed through various convenient endpoints in the cloud; not all traffic must necessarily be routed to a centralized data centre. Data partitioning models allow data to be stored as shards in many different locations to speed up retrieval. Users can access data distribution endpoints that

are created on the edge of the network, especially in highly distributed clouds such as that of Akamai Technologies, and pay only for the *last mile*. As well, the data resident on multiple devices that may be used to connect to a single account in the cloud may all be automatically and continuously synchronized.

1.2 Security Issues in Clouds

1.2.1 Security Problem Definition

Despite the economic benefits of outsourcing computation and data to the cloud, the process poses very significant risks to its users. Because user data is stored and executed on within the cloud domain, and because there is little or no external visibility into how the cloud infrastructure is implemented and managed by the provider, there is significant concern over the security and privacy of transactions and long-term storage of sensitive client data. A client has no assurance of exactly where application data and logic is stored, whether it is replicated or cached, how long it is kept for, and who exactly has access to it.

Ideally, data ought to be kept confidential not only from other clients sharing the cloud resources, but also from the cloud provider itself, using suitable data encryption techniques. Security should be enforced through technical means beyond contractual obligations between the client and provider. IT executives tend to rate security as their highest, or one of their highest, concerns in the use of cloud computing services [68]. Clients need assurance of sufficiently robust security and privacy in a cloud system before committing to it tasks that add core value to an organization and thus cannot be placed at risk. Because IT organizations are reluctant to devolve responsibility of security to a cloud computing provider, the provision of an effective security framework within the cloud is essential. "It is not recommended [for the customer to] entrust a cloud provider to manage [their encryption] keys — at least not the same provider that is handling [their] data [...] Because key management is complex and difficult for a single customer, it is even more complex and difficult for [cloud providers] to try to properly manage customers' keys." [78].

Simultaneously, cloud applications must remain accessible from a heterogeneous mix of computing devices efficiently, so that the costs of additional computation and communication do not significantly degrade the operation of the cloud and the mobile user experience. The goal of security researchers in this field is to develop techniques to ensure security in

cloud computing systems at reasonable cost. Only by overcoming these challenges, will enterprise companies increasingly migrate to the cloud to reap its economic benefits as well as open up new classes of massively scalable and useful applications.

The main scope of this thesis is the proposal of novel key management schemes to protect communication between the cloud and its users, as well as preserve the confidentiality of data stored in the cloud, in an efficient and highly scalable manner, and to thus help improve the security of cloud systems and advance their commercial potential. The proposed solutions are meant to be realistically applicable to commercial systems that are foreseeable or already in operation today.

1.2.2 Impact of Cloud Features on Security

In order to better understand the security issues present in cloud computing systems, it is instructive to first examine the high-level feature set of a typical cloud system. A cloud is essentially a distributed computing platform that can run many computational tasks in parallel. It is typically hosted in near-centralized fashion on a few large data centres that that are geographically separated for reasons of redundancy and transport efficiency. A cloud may also be implemented in a fully decentralized fashion on a large collection of interconnected peer servers. The platform, including all computational resources, are owned and operated by a cloud services provider. A number of key characteristics common and unique to clouds have been recognized [1, 79], and are summarized here. The security concerns associated with each of these features are also provided to show their relevance.

- *Virtualization.* Cloud computing can be implemented on commodity hardware and the application that runs is abstracted from the underlying hardware resources via a process known as *virtualization*; a system virtual machine allows a single server to host multiple operating system instances. A hypervisor isolates and protects individual application instances. Since it is impossible for the client to control what server a cloud application is assigned to at any given time, all hardware belonging to the cloud provider must be considered to be a monolithic and untrusted entity.

- *Resource pooling.* The cloud provider's computing resources are pooled together to serve the requests of many clients concurrently, in true multi-tenant fashion. Virtual resources are instantly and dynamically reassigned according to current demand.

Since data may be replicated for caching and redundancy reasons, to guarantee performance and safety, it may be assumed that data stored in the cloud is untraceable by the client. There is the potential for unauthorized access to confidential data that is stored on the cloud provider's servers, by another client or even the provider itself. Furthermore, client applications may be subject to interference or side-channel attack by malicious applications running simultaneously on the same server.

- *Broad network access.* Cloud application services are accessed over the Internet by a large and heterogeneous mix of clients that may vary in processing capability and in the security of the communications medium used. Desktop and server machines access the cloud through fixed high-bandwidth lines, while mobile devices are limited to the wireless medium and require greater communication efficiency. A malicious party can potentially read or manipulate data that is in transit to the cloud; thus, sensitive data requires protection when it is *in-flight*.

- *Unbounded storage and lifetime.* Data may be automatically replicated for reliability reasons and remain in storage in the cloud indefinitely, thus requiring strong encryption when it is *at-rest*. If key material required for data encryption is made accessible to the cloud provider, then it is subject to unauthorized retrieval by an honest-but-curious administrator, by an attacker, or through a legal court order.

- *Elasticity.* A central feature of the cloud is that a client may automatically scale an application to meet real-time demand; the number of running instances may be adjusted dynamically. Although it may seem that a cloud provider's resources are virtually unlimited with respect to supporting data security through cryptographic computation, this is not truly the case. Such operations are relatively very computationally expensive; the resources allocated to this overhead are at the cost of useful client application work.

"As we move to this new era of cloud computing [...] data security needs to be understood as something new, requiring new and innovative solutions [...] it is incumbent on us to generate innovations in our concepts of data security and integrity. We need tools and processes that recognize the ephemeral nature of data and the reality that physical locational controls simply will not work going forward. With a little hard work, we can achieve security models that minimize risk and enable this new method of computing. We don't need to give up on security; we simply need to abandon some of our metaphors [66]."

1.3 Overview of Contributions

This thesis makes a number of contributions to the fields of security and privacy in the context of mobile cloud computing applications; an introductory synopsis appears here:

Three novel techniques are proposed for key management that ensure the confidentiality of data outsourced to the cloud. Strategies for efficient key generation and distribution are provided to address the unique challenges of mobile cloud computing. In particular, cryptographic operations that are relatively slow to process are performed by the cloud provider without it being able to decode the encrypted data stored on its servers. In addition, the number of communication sessions required for mobile users to retrieve keys and data is minimized. A trusted third-party may be utilized to assist with the protocol if it exists, but is considered optional in most cases. Furthermore, revocation of user access is carried out in an efficient manner. The first key management scheme relies upon transformation of the encrypted data by the cloud provider; the mobile data owner does not need to constantly re-encrypt data to prevent a user that has left the system from continuing to access it. Unlike other works, users may access data directly from the cloud without an intermediary. The second scheme grants access to data based on a recipient holding correct attributes in a way that simplifies key distribution and assigns most computation to the cloud provider; in addition, it allows for optional access control through a secret group key. In the third scheme, keys are securely stored in the cloud, but disappear over time as a form of access control, while utilizing the cloud's scalable and economical storage. The proposals in the thesis are distinct from many existing works, which tend to assume that the user population utilizes powerful desktop computers that are capable of performing frequent and complex cryptographic tasks. Other works also assume that users are always connected and available, and so network usage is not of particular concern. Some works rely upon the presence of a trusted third-party within a network, which is a possible point of failure and results in added expense. Finally, other works fail to consider use cases where extremely large and constantly-changing mobile user populations are present, where the focus on efficiency in this thesis becomes highly relevant.

Additional use contexts relevant to mobile cloud computing are also considered. It is recognized that the data stored in a cloud may actually be public, or at least is required to be made readable by a cloud provider; however, it may be important to preserve the confidentiality of a user query so that the cloud provider does not learn a user's personal information based on the data accessed. Hence, a technique is presented where the cloud

provider retrieves data without essentially knowing exactly what is being retrieved. Unlike other works, it is designed to conserve the amount of information downloaded by a mobile user over a wireless network; furthermore, it does not require a trusted intermediary.

With all proposals, the algorithms have been implemented and benchmarked on actual popular mobile device hardware and a real cloud computing system. In addition, simulations have been run to assess the scalability of the schemes. The ultimate viability and performance of a software system, and its cost tradeoffs, normally becomes apparent only through actual implementation and experimentation; this facet is often missing from existing related work. Furthermore, the techniques proposed are not implementation-dependent; the algorithms and libraries that they depend on are highly portable or adaptable to most of today's commercial mobile and cloud platforms.

A detailed comparison of related work is found in Chapter 3 on page 30, and a more detailed list of contributions is found in Section 8.2 in Chapter 8 on page 148.

1.4 Outline of Thesis

The thesis contained herein addresses communication and data security in the context of mobile cloud computing. In Chapter 2, a model of a cloud computing system and its mobile users is presented, with a discussion of possible security threats and factors to consider in securing such a system. Key management techniques that permit secure data storage in the cloud are then presented. In Chapter 3, related work in academic literature, as well as in current commercial solutions, is summarized. In Chapter 4, a model for key management utilizing data re-encryption in the cloud is proposed. In Chapter 5, a hybrid model for key management combining attribute-based encryption and data re-encryption in the cloud is proposed. In Chapter 6, a key management model utilizing cloud-hosted key sharing is presented, which takes advantage of the cloud provider's scalable storage rather than computation. Additional complementary aspects of information exchange in mobile cloud computing systems appear next. In Chapter 7, a technique for private information retrieval from a cloud is proposed, where the stored data is public but the client's queries are kept private. Finally, in Chapter 8, a summary of research contributions from all chapters is presented and future work is proposed. In Appendix A, projected cryptographic computation and communication costs for mobile device users and clouds are suggested as a thought exercise.

Chapter 2

Background

The focus of the thesis is on the area of communications and data security for mobile cloud computing systems. Various ways to organize clouds are depicted in Section 2.1. A discussion of the architecture of a cloud system is presented so that the components that require protection may be understood. An overview of the internal operation of a mobile-based cloud and its role within a system and network model appears in Section 2.2. Typical applications and use cases are suggested in Section 2.3 to show how a contemporary cloud system is utilized and to aid in identification of features that require security protection. A threat model is demonstrated in Section 2.4 that captures the potential points of attack in a cloud system. The criteria that an ideal security mechanism ought to satisfy are elaborated in a list of design considerations in Section 2.5; a comparison is made against the factors associated with other kinds of systems such as grid computing, to show the unique constraints of a cloud. In particular, the key management function is emphasized as being the most relevant to the security problem under study. Finally, guidelines for assessing the usefulness of a security solution are suggested in Section 2.6.

2.1 Cloud Organization

2.1.1 Division of Resources

Clouds may be organized in various ways with respect to the division of responsibility between the client organization and the cloud provider:

- *Public cloud.* Applications are all hosted on infrastructure controlled by the cloud provider. The services or computational results provided by these applications are accessed over the public Internet and optionally over wireless networks. Communication must be performed over a secure channel to ensure confidentiality.

- *Private cloud.* The efficiencies of cloud computing are realized on an infrastructure that is on-premise, or internal to the organization. This option is applicable to an enterprise with an existing investment in a distributed system that wishes to retain greater control over its proprietary data and administration. It is particularly suitable for mission- and business-critical applications. The client can leverage existing infrastructure to avoid the cost of migration to a public cloud and its subsequent operation. The underlying rationale is that these customers may also be subject to laws that restrict the location of data due to its jurisdiction, or may simply be concerned about running highly sensitive applications on a third party's servers. This option limits the potential for runtime scalability, however. In a private cloud, a client may access the application over a protected intranet or via a secure tunnel.

- *Hybrid cloud.* The above two approaches may be combined in a hybrid solution that connects a company's internal infrastructure to the cloud provider's, resulting in a cloud partitioned into private and public components. This is sometimes known as *IT infrastructure bridging.* A client's private network may be shielded from a cloud service provider's public cloud network by a firewall. One advantage is flexibility in the assignment of data collections or tasks between a private domain that is more secure and a public domain that is more scalable. In terms of network architecture, hybrid clouds may be decomposed into cluster controllers that manage the network traffic to individual nodes making up the cloud. Each cluster is considered to be a private network, and a central cloud controller manages all clusters.

2.1.2 Network Organization

Cloud architectures can be categorized according to the distribution of back-end resources:

- *Centralized cloud.* A cloud provider may operate multiple data centres, which are adequate when application users are within the same geographical region as one of the host centres. Data centres are generally separated for continuance of operations in case of natural or man-made disasters. Additionally, the availability of multiple connected data centres can reduce latencies for a global user population.

- *Highly-distributed cloud.* A distributed approach entails a great number of servers being hosted in many locations and on many networks worldwide. Application instances can be automatically created in certain regions based on real-time demand. The location of an application component depends on its function. Content-serving application components can be built at the edge of the cloud to reduce latency, while transaction-oriented components requiring consistency can be run at the origin infrastructure. Authentication functions may be performed at either end [69].

- *Grid computing.* Although not strictly a variant of today's definition of a cloud system, it is worthwhile to consider a pre-cursor that links disparate computers to form one large infrastructure. Essentially, a supercomputer is formed from a cluster of networked, loosely coupled computers working together to perform very large tasks. The grid infrastructure allows service-oriented, flexible, and seamless sharing of a heterogeneous network of resources for intensive tasks. The goals are to provide faster throughput and higher scalability, while keeping costs low [22]; this loose organization differs from that of cloud computing. Grid computing eventually evolved into cluster computing, where computing resources are connected together by high-speed interconnects; and peer-to-peer networks, in which peers directly share information. In contrast, the internal organization of a cloud system is opaque to its clients. Cloud computing further offers virtualization, web service technology, and programmable interfaces as differentiating technical features.

2.2 Cloud System Model

2.2.1 Internal Cloud Architecture

Broadly speaking, the chief design goals of a cloud computing system are to provide an environment to allow scalability of application performance and efficient virtualization. Key administrative functions that also need to be fulfilled are performance monitoring and management to reliably allow for various levels of service. The architecture of a cloud computing system typically contains a set of layers that are common to most providers, as illustrated in Figure 2.1; the model shown is derived from cloud reference architectures proposed by IBM and Oracle [2,14]. The following is a functional description of the layers, from the topmost to the lowermost layer:

- Application services provide business logic, web hosting, and interactive features delivered over a wired Internet connection.
- A management services layer is responsible for managing the underlying hardware resources that fulfill application processing and storage demands. It includes application template storage and provisioning, application performance monitoring and dynamic workload management, security policy management, and billing functions.
- A cloud infrastructure fabric includes virtualized physical servers, network connectivity, and storage. Resources are allocated by the upper layer to execute applications consistent with service level agreements through optimal workload management.

To deploy a cloud application, a client typically creates an image and uploads it to the provider. The image is then instantiated and replicated automatically within the cloud to satisfy the processing demands of users at any given time. This process is called *provisioning*; the provisioning service allocates processes among available data centres and ensures rapid reconstitution of services if necessary.

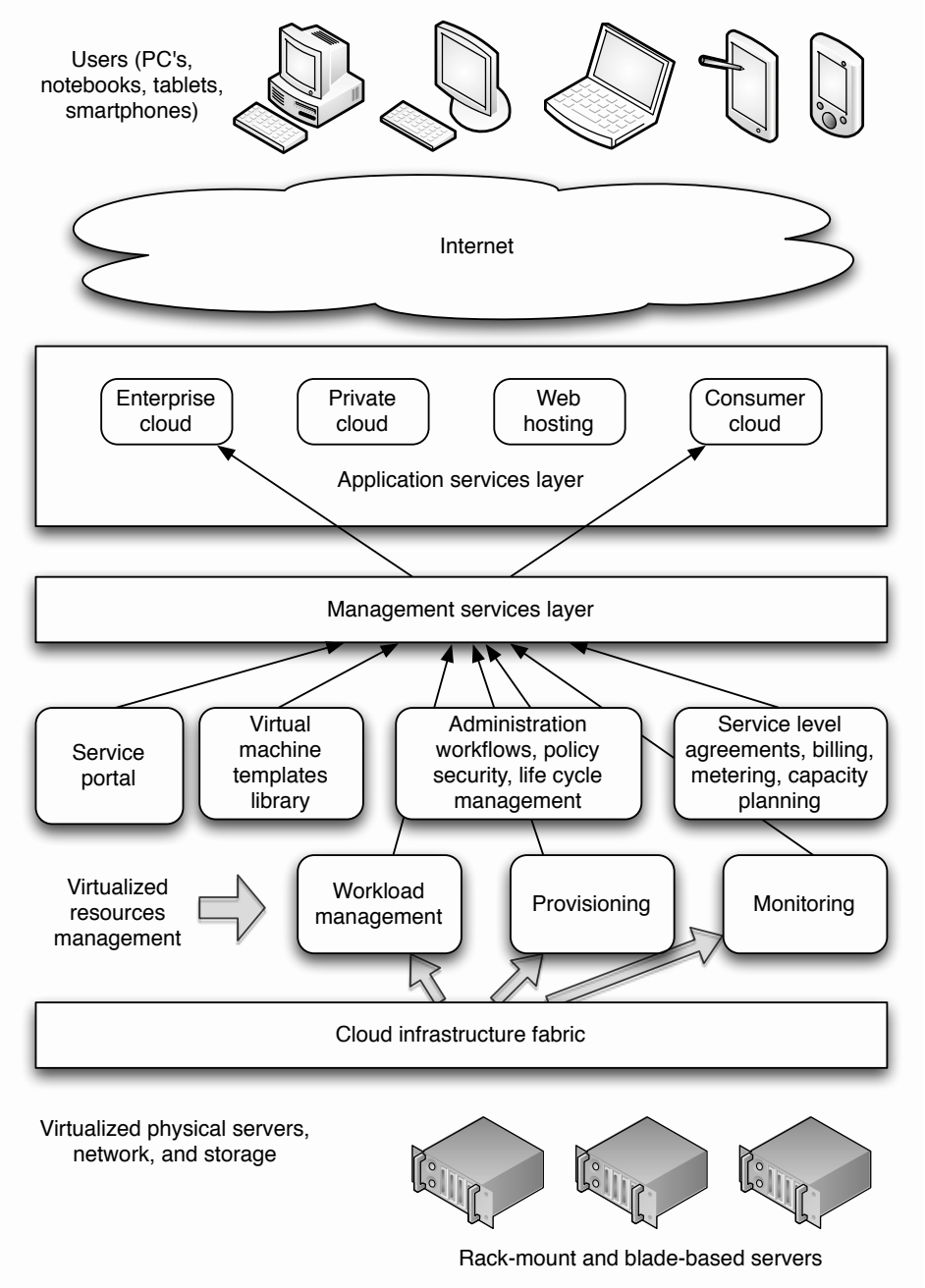


Figure 2.1: The internal architecture of a cloud.

2.2.2 Network System Model

A complementary external system model of a cloud is illustrated in Figure 2.2. It consists of multiple data centres that are administered by a central controller, which manages requests that arrive at external interfaces in the cloud's firewall. Client administrators have control over port access for their applications.

Requests are made over the public Internet, which is considered a normally reliable but insecure medium. Internet traffic is routed through a topology of network switches, which culminate in individual Internet Service Provider (ISP) network switches connected by high-capacity optical links such as OC-3; this packet data network may be bridged to a wireless 3G or 4G infrastructure through a gateway node, allowing smartphones to connect wirelessly to 3G or 4G towers and switches. Additionally, smartphones may communicate among themselves using short-range local links such as Bluetooth. Another entry point into the Internet is via a router and Wi-Fi access point, enabling notebooks and wireless sensors to connect via some Wi-Fi standard, such as 802.11 or low-cost 802.15.4 as part of a Personal Area Network. Desktop computers may connect via Ethernet to a router directly, which typically integrates a firewall function.

2.3 Applications of Mobile Cloud Computing

A security framework incorporating secure data storage within the cloud and secure communication channels has numerous practical applications. If the solution is highly efficient and scalable, then it may be more readily utilized by resource-constrained devices. These devices may interact with clouds in a variety of different applications to solve real world problems, such as document and media storage, user collaboration, and data analytics. Mobile device industry trends and use cases are captured in the following discussion.

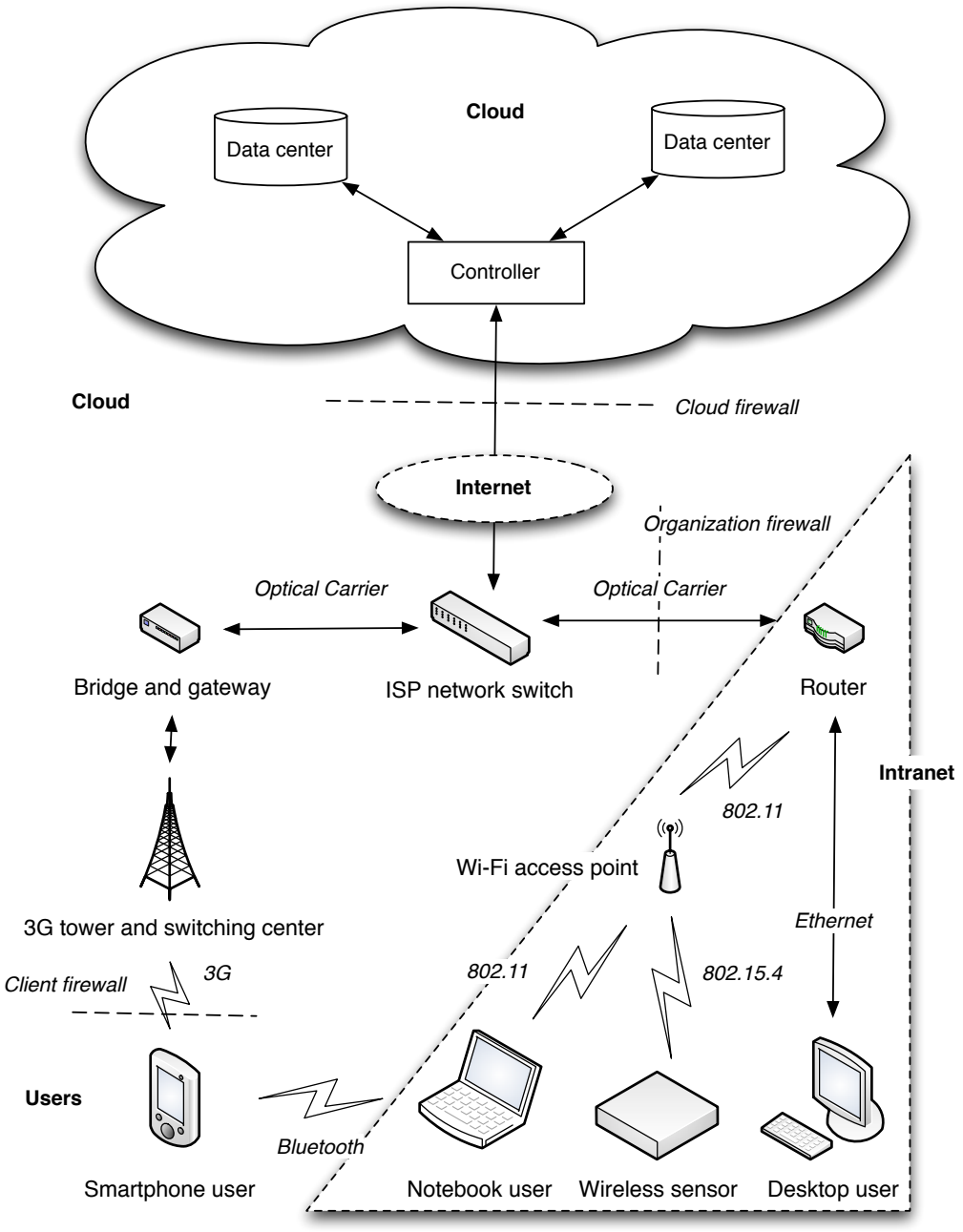


Figure 2.2: A network system model of mobile cloud computing.

2.3.1 Use of Resource-Constrained Devices

Mobile- and Sensor-Based Computing

There has been a very evident trend towards the adoption of smartphone and tablet devices that enable users to run complex applications interacting with cloud services, including media repositories such as music streaming, and always-on collaborative services such as social networks [55, 59]. It is reasonable to predict that users will increasingly require persistent access to cloud applications from a variety of highly mobile machines. Connections are typically made using standard wireless packet data protocols with TCP/IP as the transport; their use is relatively expensive. Just as importantly, mobile devices have limited memory, battery life, processing performance, connectivity, and available bandwidth compared to their desktop counterparts, all of which complicate protocol design.

Numerous cloud applications exist today, or are being actively researched, that cater to the resource demands of smartphone users. By offloading computation of complex tasks to the cloud and storing associated data in the cloud, mobile device users may enjoy numerous advantages; such a framework serves to extend onboard battery life, conserve local memory, speed up processing of tasks, and improve reliability by storing data in a centralized manner. Various application areas and platforms are being investigated by researchers [30], including in the domain of mobile commerce, mobile learning, collaboration, health care, gaming, and web searching. Commonly used artifacts such as enterprise and personal e-mail, movies and music, and various other forms of user data are already outsourced to the cloud today; reasons for doing so include greater storage capacity, expanded options for sharing, leveraging network effects, and data safekeeping.

The wireless sensor is another important class of device that is not necessarily mobile, but may also utilize a cloud and wireless network, and is even more resource-constrained than the devices already mentioned; hence, it is considered relevant to the discussion. For instance, Sensing Planet's cloud-based platform provides real-time online instrumentation, management, and control of a connected wireless sensor network. It uses the ZigBee 2.4 GHz radio frequency, based on the IEEE 802.15.4 standard for LRWPANs (Low-Rate Wireless Personal Area Networks). Microsoft's Azure cloud platform integrates with Living PlanIt's Urban Operating System, a real-time urban control platform converging cloud computing with the fabric of buildings containing embedded sensors [81].

Mobile Device Classification

Multiple classes of devices are now identified to define the capabilities of the user class in a mobile-based cloud system, with typical recent-generation specifications listed in Table 2.1. In the case of wireless sensors, typical characteristics are assumed [75] of devices with processors running wireless data-driven Time Division Multiple Access (TDMA) applications on TinyOS, a prevalent sensor operating system.

Criterion	**Smartphones**	**Wireless sensors**
Processor	Single/dual-core, 0.6-1 GHz	12 MHz
Memory	256-512 MB	2 kB
Mobility	Highly mobile	Fixed (but possibly mobile)
Battery capacity	1200 mAh	240 mAh (per cm^3)
Power	Up to 1200 mAh per day	8 mAh per day
Wireless connectivity	Transient and intermittent	Constant and reliable
Wireless data rate	2800 kbps download (3G)	20-250 kbps (IEEE 802.15.4)

Table 2.1: Example specifications of the classes of user devices studied.

2.3.2 Use Cases in Mobile Cloud Computing

Numerous practical use cases may be described where data is securely stored in the cloud and accessed, which a security framework should be designed to accommodate. The user base consists of mobile device users that regularly upload content to the cloud that is then shared with other authorized users. For example, the data may consist of customer records in the case of an enterprise application, or personal photos and videos in the case of a consumer-oriented personal productivity and entertainment application. These applications entail two-way data exchange between users and the cloud. One-way data exchange may also be envisioned; for instance, data may be stored in a relational database or in a log-based file format, which may be applicable for analytics purposes.

The user membership may include a workplace department or a collection of social friends; it is considered to be dynamic and constantly evolving, with users joining and leaving in a frequent and unpredictable manner. Each member of an authorized user group is deemed to operate at an equivalent trust level, although each data record may

require its own unique access permission that may be shared by any set of users. Thus, there is a many-to-many association between data records and users in terms of access rights. For instance, a group of senior managers may have access to a collection of records, a subset of which may also be accessible to the rank and file. In terms of the network model, each user group will obtain access to a collection of linked and related records [101] in the cloud called a data *partition*.

The permanent cloud data store may be accessed through a *key-value* mechanism, in which a valid key index must be supplied to retrieve the value stored at the index location. In each communication session with the cloud, a user may send a data storage or a fetch request, identifying the data record with a unique numeric record identifier, and similarly, the partition with which is it associated with. The identifiers, but not the data content, are deemed to be public knowledge. Repeated access of the same records in the cloud by the same user is to be anticipated, as local storage on a mobile device is very limited.

Specific security features may be identified to satisfy these use cases:

- *Forward secrecy.* A previously authorized user that leaves an authorized user set, in a process known as *revocation*, should no longer retain access to encrypted data. For example, a consultant may require only temporary access to company resources for the duration of a project, or an employee may permanently leave a firm.

- *Backward secrecy.* A user that joins an authorized user set may also not necessarily obtain access to resources that were made available before the join occurred. This rule is of lesser practical value, but may be useful in special cases such as mitigating the risk of wilful patent infringement by limiting the sphere of known information related to intellectual property.

- *Blind storage.* Data stored in the cloud remains encrypted at rest to prevent the cloud provider from gleaning confidential user information at any time.

These features also serve to limit the amount of useful information that may be gleaned by an attacker that successfully obtains valid key material in some instance of time.

Note that an underlying assumption of this work is that mobile users are in constant connectivity with the cloud via a 3G or 4G wireless network or a Wi-Fi access point; however, this assumption may not hold in the following cases: temporary connection loss

suffered by a user when indoors, a user being exposed to excessive wireless data usage fees as a result of international roaming, and the need to transfer an extremely large amount of data such as an operating system update; in such cases, it is useful to determine whether direct peer-to-peer links between mobile devices may be exploited to continue sharing content without direct participation of the cloud provider. Because the devices must be in very close proximity, data confidentiality is less of a practical concern than efficient sustained transfer. A high-level framework for peer-to-peer file transfer is proposed in [112] that specifically addresses the resource constraints of mobile users. A mobile user with fast connectivity is elected as a super-node, creates an initial seed from data stored in the cloud, then distributes it to other users that subscribe their interests to that content. Experiments on smartphones are conducted to determine how throughput is improved by dynamically controlling variables such as file segment sizes in a communication protocol utilizing Bluetooth. A parameter such as a target upload-to-download ratio may be dependent upon the dynamic state of a device, such as its current battery level.

2.4 Cloud Computing Trust Issues and Threats

2.4.1 Adversary and Threat Model

The cloud provider is expected to store and retrieve data, to and from its permanent data store, upon request by users. The cloud provider is situated in a network domain outside of that of the users, and its internal operations are opaque. The cloud provider, including all parties within its scope of operations such as an internal administrator, is assumed to be an *honest-but-curious* adversary without malicious intent; it will obey a communications protocol and application logic deployed by the client, and will not deny service to any authorized party or cause other interference. Furthermore, it will provide reliable service to users, including the provision of persistent storage capacity on demand, and data replication to the extent that it is paid for; it will also honour all data upload and download requests. At the same time, an administrator of the cloud or any other party that may gain access to the cloud storage cannot be trusted to not read, copy, and retain confidential data that is stored within the cloud for nefarious reasons or simply out of curiosity; such access can occur without the client's knowledge. If the data is stored in encrypted form, then any party that gains access to the storage and key material may be

expected to decrypt and read the confidential data contents.

It is not expected that the cloud provider will attempt to create, modify, or delete data in a data partition that is understood to belong to an external client or a set of clients; nor will it serve incorrect or modified data to a user upon request. As a result, no facility is required to detect that tampering of data has occurred. If the cloud provider is not trustworthy to the described extent, then additional safeguards would be needed to verify operations and the integrity of data, and corresponding performance overheads would also be incurred; such features are outside the scope of the work. Nevertheless, the honest-but-curious characteristic is deemed reasonable and realistic in view of the degree of trustworthiness of recognized public clouds currently in operation.

As part of its normal activities, a cloud provider may replicate any encrypted data with or without the user's consent, but this should not aid the provider or an external unauthorized party in gleaning any additional information helpful to decrypting the data stored in the cloud. It is unlikely but conceivable that a provider may collude with a rogue user or another provider, if one is available and contains relevant information, to defeat a security mechanism and satisfy the curiosity of either party as to the contents of cloud data. Collusion is to be prevented if possible, but is a lesser concern overall. Attacks may also originate within the cloud itself from other tenant clients of the cloud servers. The goal of this thesis, however, is to address data and communication security that is external to the cloud implementation; for instance, hardware security, protection of running instances and inter-instance communication security is outside its scope.

From the perspective of network security, the cloud itself is accessed over an open Internet infrastructure, which is not considered to be highly secure. It may be bridged to a wireless infrastructure consisting of an air link that is also considered insecure. This scenario is borne out in practice: for instance, the original GSM telecommunications system relies upon a very limited 64-bit A5/1 stream cipher; although it has since been replaced by the 128-bit A5/3 cipher on 3G networks, its effective length is still only 64 bits. The A5/1 encryption has been successfully attacked by creating a *rainbow table*, in which the encryption key is reconstructed in a real-time attack [86]. The Code Division Multiple Access (CDMA, standardized as IS-95) system also uses a very limited cipher; its security relies on a pseudorandom so-called *long code* that is only 42 bits in length. It has been shown that an eavesdropper can recover the required code after eavesdropping a transmission on the traffic channel for about one second [122]. Clearly, some form of strong encryption for

the communications channel between the mobile device and the cloud system is required to ensure the confidentiality of sensitive data exchange.

Users, whether connecting from desktop or mobile devices, implicitly trust their local machine. That is, when data is reliably retrieved from a cloud server in a correct state, the user's machine is trusted to perform the necessary cryptographic operations to retrieve and display the original plaintext. Authorized users with common access to a data share are considered to belong to the same organization or community, and hence may share key material without compromising the security of the entire system. If a user chooses to share key material with an unauthorized user, such an attempt is not prevented, as the user could simply share the decrypted content with equal ease; the use of digital rights management, as a remedy, is not contemplated within the scope of this work.

In a variation of the system model, a manager acting as a trusted entity is present within the logical client domain and maintains a list of authorized users. The manager may be located behind the organization's firewall and is inaccessible by the cloud provider. It may perform some key management duties and enforcement of access rights, and requires high availability. It may exist within the context of a private cloud and is considered trusted by the authorized user population. However, as the manager represents a possible point of attack in itself, it is advantageous to limit sharing of private key material between users and the manager, unless it is warranted by the scheme. Collusion between the manager as a trusted third party and any user is not deemed to be a particular threat, as the manager and all its authorized users are expected to belong to the same client organization.

Once a user's access rights are revoked, any valid key information in the user's possession may continue to provide access to encrypted user data. However, this apparent vulnerability is deemed to be only temporary in nature; in practice, mobile users have limited storage capacity and are unable to cache copious amounts of data, including numerous key materials, for extended periods of time. This is especially true of a data storage system consisting of fine-grained access, where even individual data records may be encrypted with unique keys, and the storage of key material itself is onerous.

It is assumed that all mobile devices are protected against outside attack through sufficient computer security and that users cannot become compromised; other techniques related to computer security are required to ensure that secret information is not divulged between a mobile user and an outside attacker. Even if such an attack occurs, then the information illicitly gained is understood to eventually become stale and unusable.

Vulnerabilities in the cloud may be exploited to access information in the cloud without authorization to do so; typical examples are insufficient access control enforcement, unencrypted data storage and transmission, and unrestricted modification of security policies [90]. Cloud providers may not be subject to external audits and security certifications, so that the onus of keeping data secure is ultimately on the customer; the shared environment in which data resides means that data must be correctly segregated and encrypted [20]. Despite all these risks, any role that the cloud provider plays in executing a security scheme is assumed to impart the same level of confidence in robustness as its other services, and does not really constitute a single point of failure. A cloud by design is typically engineered as a distributed system with data replication, reliable servers, multiple endpoints, and other safeguards that virtually guarantee its continuous operation.

In certain commercial environments, a single organization may exert control over multiple entities in the system model. For instance, the Nexus One and Galaxy Note II smartphones used in the experiments in this thesis run the Android Linux-based operating system developed by Google. The same corporation develops and operates the App Engine PaaS cloud platform, also part of the experiments. It is assumed that Google, or any other entity under similar circumstances, will not embed any logic in the smartphone software to defeat the proposed security mechanisms, such as sharing key material or plaintext with the cloud provider. Likewise, it is also assumed that a telecommunications carrier that provisions the mobile device or allows it to operate on its network will not install any software to compromise the security of the system, such as intercepting message traffic.

In summary, the parties contributing to an attack may include an eavesdropper located along the open Internet path to the cloud, a user whose access has been revoked yet retains key information, a user belonging to the client organization but of insufficient clearance to access all of the data belonging to the client, and the administrator of the cloud system with unrestricted access to cloud resources. To minimize the privacy risk for content, it is evident that data must be encrypted *in-transit* to and from the cloud, and *at-rest* in the cloud. The fact that the shared environment of a cloud has intrinsic vulnerabilities and limited oversight suggests that many of the concerns may be alleviated by assigning management of security keys to a client or a trusted third-party, outside of the domain of a cloud provider. If the client controls most aspects of security, the risk and impact of third-party negligence is mitigated.

2.4.2 Security Issues in Grid Computing

It is instructive to examine the security issues inherent in grid computing [32], as a relevant pre-cursor to cloud computing:

1. Protection from external threats must allow the sharing of resources across organizational boundaries, for which traditional firewalls are unsuitable.
2. Trust relations are agreed upon at the organizational level rather than the user level.
3. Grid nodes are dynamic and unpredictable in nature. The efficient updating of group keys is a particular problem.
4. Grid computing systems are distributed and heterogeneous. Centralized authentication is generally unavailable.

From a cloud perspective, the implication is that authentication should occur at the client's organizational level, and that it must be efficient for dynamically changing groups. Although a cloud system can offer centralized authentication, its high scalability does result in challenges in implementing efficienct key management, as it does in grid computing.

Some of the stated concerns originally found in grid computing can be addressed by group keying. Grid computers may involve resources that are shared across the Internet, and point-to-point communication using cryptographic schemes is inefficient; broadcasting a group key may be an accepted alternative. Groups may be dynamic and can be organized in real-time according to available resources. Broadcasting is of limited value in a cloud application, however; due to the majority of network communication being over the Internet and wireless network, it could become prohibitively expensive, especially for mobile users; the cloud provider typically bills the client for such downloads, and wireless transmission incurs its own usage and energy costs.

2.5 Design Factors for Cloud Security

The lack of direct control over cloud resources forces clients to either fully trust the cloud provider, or adopt cryptographic protocols to ensure the confidentiality of stored data. The use cases discussed above exert influence on the appropriate design of a secure solution. The following general guidelines are identified, the focus being on communication security and encrypted file storage in particular:

2.5.1 General Security Features

1. *Encryption.* The fact that the shared environment of a cloud has intrinsic vulnerabilities and limited oversight suggests that data must be permanently stored within the cloud in encrypted form, with keys being controlled by the client or a trusted third-party. End-to-end encryption must be offered for all data communication, as it traverses the open Internet. Users typically interact with cloud applications via sessions, and perform regular updates to data, suggesting the need for session management and data versioning. The multi-tenant nature of cloud computing requires isolation between the individual users of a cloud application.

2. *Scalability.* Many devices may be connected to a cloud application simultaneously, and all sessions must be individually protected. The multi-user environment may potentially scale to many thousands or even millions of users. In an extreme example, the social network Facebook has over 400 million active users, half of whom log into the system in any given day and spend about an hour on the site, on average [40]. In an enterprise system, users may be created or removed at great frequency, and communication costs related to key management must scale accordingly.

3. *Access control.* Using appropriate credentials, the user must authenticate with the cloud network before being granted access to key material. Users of different class privileges may exist, and so it is advantageous to provide prioritized or hierarchical access control in some cases. Furthermore, the data owner may be mobile and not always available to fulfill the administration duties of these functions.

4. *Data partitioning.* Members belonging to a subpopulation of all users will typically require access to a common data partition resident in the cloud storage system.

Thus, it is desirable for data permanently stored inside the cloud to be segmented into addressable data partitions, such that appropriate access rights are enforced on each. Clients may even require fine-grained security controls on the record-level.

5. *Data lifetime.* User data stored inside the cloud is typically replicated and archived; it may exist for an indefinite time. If key material accessible by the cloud provider is not destroyed once the data is no longer needed, it may be retrieved in the future by a malicious party or through legal means such as a court order. Some data, however, requires confidentiality protection for only a limited amount of time, after which the security of the data, or the data itself, becomes irrelevant; an example is a financial market transaction.

6. *User diversity.* A contemporary user may connect with the cloud from any one of a heterogeneous mix of user devices. In some cases, authorized access may need to be associated with a particular user, rather than a particular device. This concept may involve a reunion of user information across multiple device accounts in a single identity management system.

7. *Mobile access.* Mobile devices are exposed to a number of resource challenges, network rate limitations, and intermittent connectivity. Communication and processing requirements of a secure protocol must be as limited as possible, without significantly compromising its performance and efficacy.

8. *Efficient server transactions.* Any cryptographic protocol must be lightweight in computation and key updates must be relatively small and infrequent; otherwise, a cloud provider may be ill-equipped to handle this traffic, leading to an outage [105].

9. *Hybrid architectures.* A scheme for communication security must be compatible with popular computing models. Cloud computing is evolving to support hybrid systems where secure communication between a private and public cloud is assured, and where computation may be distributed between private and public realms.

2.5.2 Key Management Functions

From a functional point of view, a candidate key management solution for a cloud computing system will encompass most of the following operations and properties, some of which are referred to in [80] [92]:

1. *Trust.* The user must obtain trusted credentials that permit access to the cloud. The cloud must recognize the user as belonging to a trusted Access Control List (ACL), or the user must be found to be trusted by the other members of the group.

2. *Authentication.* Using appropriate credentials, a user will authenticate with the cloud network before being granted access to key material. The user must be provisioned with a secret key from the cloud itself, a trusted third-party entity, or other trusted users. Key material can be confirmed to be valid for a particular user, and it must be kept confidential and secure from attack.

3. *Encryption.* The user must utilize a key to secure communication with the cloud, and to secure the data that is stored within the cloud and accessed by other users. Due to the longevity of cloud data, appropriate key sizes must be chosen. NIST (National Institute of Standards and Technology) recommendations [13] for comparable security strengths beyond the year 2030 include: 128 bits for symmetric algorithms (e.g. AES), 3072 bits for asymmetric algorithms (e.g. RSA), key and group sizes of 256 and 3072 bits, respectively, for discrete logarithm algorithms (e.g. Diffie-Hellman), and 256 bits for elliptical curve cryptography (e.g. ECDSA).

4. *Refresh.* The key that the user holds must be updated periodically to limit the opportunity window for an attacker to decrypt communication with the cloud, and to minimize the useful information obtained if the attack is successful.

5. *Revocation.* The user, upon leaving the cloud network, must not continue to retain a valid key that can be used to access the cloud or decrypt the communication of other users. The key must be destroyed or become obsolete. If access rights are revoked, then the user can no longer remain authenticated with the network.

6. *Availability.* Key management services must always be available to users to ensure uninterrupted communication and continuity of cloud services.

7. *Scalability.* Key management services must efficiently consume, directly or indirectly, scarce resources such as wireless bandwidth and energy. In so doing, they must support high scalability of the user base without degrading the level of service.

2.5.3 Comparison to Other Systems

The security issues noted here are fairly unique to the problem of mobile cloud computing. They differ from more traditional key distribution schemes found in grid computing, client-server computing, and peer-to-peer networks in the following ways:

1. Most cloud systems consist of centralized data centres that users must access directly. The existence of an infrastructure can be relied upon, for centralized key management or other purposes. Peer-to-peer communication over a wireless network is possible but it may be difficult to find available peers and in close proximity if required.

2. The user base consists of a heterogeneous mix of devices, many of which are mobile devices with tight resource constraints. They may include devices ranging from multi-core tablets to RF sensors. In contrast, desktop users have significantly greater computational power and storage capacity. For instance, an SSL handshake on a notebook (with a Pentium M 1.86 GB CPU) was found to take only 31% of the time that a smartphone (with a 624 MHz PXA270 CPU) took to finish it [100]. Therefore, a cryptographic scheme that is *asymmetric* in nature may be appropriate in mobile cloud computing: the amount of cryptographic computation on the mobile device is minimized to improve responsiveness of the user interface and to limit battery drain, while a cloud server can take advantage of its inherent scaling property to carry out cryptographic work by commissioning additional application instances as needed.

3. The cloud provider must be considered to be a non-trusted entity. Any key material stored in the cloud must be protected not only from outside access, but also from cloud administrators, to ensure maximum data confidentiality. Alternatively, keys may need to be stored outside of the untrusted cloud domain. Note that a key management server, whether it is located in the cloud or off-site, may store various kinds of keys, including authentication keys, authorization tokens, file encryption keys, hardware storage keys, and certificates.

4. Within an enterprise network, the user can rely on multiple layers of authentication including password-protected logins. In a cloud system, users may not be verifiable against a corporate directory, as it is normally located behind the client's own firewall. Users may be widely distributed and not utilize Virtual Private Network (VPN) tunnels, complicating access management.

The stated requirements address the unique properties of cloud computing systems; they do not necessarily pertain to general-purpose web servers which are more fixed in resources. For instance, a web server may not be able to seamlessly and quickly scale in terms of its processes, memory, and network bandwidth, which affects both application operation and key management. A standard web server will also typically be physically located in one place, whereas cloud systems typically operate multiple data centres to reduce latency for global services, and thus network topology is less relevant. Finally, few web servers are equipped to handle the scale of users being examined in this work. For those enterprise servers that do, their back-end typically mirrors the characteristics of private cloud systems such as blade system hardware setups, application template creation and provisioning layers, and distributed workload management.

2.6 Performance Assessment Criteria

Key management systems may be evaluated not only in terms of compliance with the security features already outlined, but also in terms of their performance in various areas such as resource usage. Although some schemes may perform well even on resource-limited mobile devices in small numbers, it may be impossible to attain extreme scalability in a cloud computing scenario. Some of the following quantitative performance aspects will aid in evaluating various approaches to key management:

- The amount of storage required for key material on the server and on mobile devices; the latter is especially restricted due to limited onboard flash memory.
- The amount of data that must be exchanged between parties to carry out cryptographic transactions. A wireless communications medium has limited capacity and carries with it a high usage cost; it should therefore be minimized.

- The amount of computation required, particularly on the mobile device, to carry out the encryption and decryption operations.
- The amount of energy consumption on the mobile device due to wireless data being transmitted and received, or due to prolonged computation occurring.
- The operational memory footprint required on the mobile device; this includes dynamic RAM (Random-Access Memory) used for temporary storage and flash memory used for permanent storage.
- The economic cost for the client of the cloud service, based on the total amount of cloud computation required, and the amount of Internet communication consumed by the cloud application and charged to the user.

The following commentary is useful for motivating the study of performance: "We're in great need of secure computation outsourcing mechanisms to protect sensitive workload information [...] This task is difficult, however, due to several challenges [...] First, such a mechanism must be practically feasible in terms of computational complexity. Otherwise, either the user's cost can become prohibitively huge, or the cloud might not be able to complete the outsourced computations in a reasonable amount of time. Second, it must provide sound security guarantees without restricting system assumptions [including] practical performance. Third, this mechanism must enable substantial computational savings at the user side compared to the amount of effort required to solve a problem locally. Otherwise, users have no reason to outsource computation to the cloud. [94]."

In Appendix A on Page 152, sample calculations are provided of the kind of energy consumption on mobile devices and cryptographic workloads on cloud computing servers that may be expected from a key management solution; the discussion provides insight into the significance of these costs in a real system.

Chapter 3

Related Work on Key Management

Recent research literature on key management techniques with relevance to cloud computing systems is surveyed in Section 3.1. Included are generic centralized and decentralized techniques for key control, as well as solutions specific to encrypted storage based on proxy re-encryption, encrypted network file systems, and more recent work on secure cloud storage. Analyses of strengths and weaknesses are provided, and opportunities for improvement as dictated by the demands of a scalable cloud system are highlighted. The related work in this chapter covers only a common baseline; it forms a starting point for the proposals in subsequent chapters, where additional related work is presented where appropriate based on the specific techniques used. As a reference point, features of the rapidly evolving commercial cloud systems of today are also presented in Section 3.2. Concluding remarks are made in Section 3.3.

3.1 Body of Academic Literature

3.1.1 Public Key Encryption

Numerous solutions may be envisaged to exchange encrypted data with a cloud provider in a secure manner such that the cloud provider is not directly entrusted with key material, but naïve schemes often prove difficult to scale. In the classic centralized model, a single authority present within the cloud computing infrastructure is responsible for access control; this notion seems consistent with the centralized nature of cloud servers. *Public key*

encryption (PKE) may be employed, in which the authority generates private keys and distributes them to individual users through secure means. The corresponding public key of a data partition is freely made available for download by the cloud provider. The underlying cryptographic method may be RSA, for instance. *Digital certificate management* can be used to verify the identity of the originator of the ciphertext. Communication is necessarily one-to-one; the originating user encrypts data with the recipient user's public key, uploads it to the cloud, and the recipient retrieves it and decodes the plaintext using his or her own private key. This general scenario of solely using PKE, however, is unrealistic when a very large user population is present; it requires that the data owner provide an encrypted version of data for each recipient that may access it, and furthermore, it requires constant availability of the owner. Likewise, a straightforward approach employing PGP encryption [125] would encounter challenges with scalability; the symmetric key used for encryption of user data would need to be encoded with the public key of each recipient. Rather, it is preferable for a data owner to perform a one-time encryption. Alternatively, if all user data is encrypted with a single key, then that key must be shared with all authorized users, which carries a high traffic cost especially if this obligation rests on a mobile data owner. Revocation would require some form of authentication to prevent access; the enforcement of it would require trust in the provider or further burden the data owner.

A remedy is to encrypt data with a *group key*, instead, and share it among a population of authorized users; the main challenge relates to distributing the keys in a secure and scalable manner without requiring the cloud provider to be trusted to manage and deploy all private user keys itself. Another important problem lies with user *revocation*; if a user leaves the authorized user set, the group key must be replaced and redeployed, a process known as *re-keying*, which scales in cost with the number of users. Also, public-key certificates must be generated by the cloud authority and deployed to all users before communication can occur. The authorization server in the cloud may become overloaded as a result of this responsibility, and potentially stop operation of the cloud. Furthermore, users may join and leave the authorized user set frequently, leading to constant key regeneration and re-distribution through additional communication sessions to handle user revocation; in a highly scalable system composed of thousands of users, such events may occur at relatively high frequency. Wireless communication, however, is expensive and results in rapid battery drain, particularly when transmitting from a mobile device [11].

Security enforcement based on the monitoring of user behaviour can mitigate these performance concerns, at the cost of reduced security. For instance, in TrustCube, a

star-shaped, or centralized, authentication system is used [30]; it provides implicit authentication by monitoring user behaviour, and it falls back to another authentication method such as OpenID if a user violates policy norms. In this case, however, the cloud provider must be entrusted with all aspects of this system, including the use of aggregated data on user contexts and activities, thus relaxing the trust model to an even greater extent.

One variant of centralized key management is baseline *broadcast encryption*, in which the key manager generates symmetric keys for multiple users. Each time that new encrypted data becomes available in the cloud, it may be broadcasted in encrypted form to all interested users, with each outgoing message being encrypted with a different key corresponding to the recipient. If the user membership changes, then new keys must be broadcast to all users, which is an unrealistic proposition in a highly scalable system. Also, the broadcast itself is an expensive use of bandwidth. To reduce the cost, a stateful scheme such as Logical Key Hierarchy [116] may be employed, in which a directed acyclic graph of encryption keys is constructed, and each user is associated with a leaf node. Whenever a user joins or leaves, rekeying messages are transmitted along the path from the root node to the user's leaf node location. In such a rekeying strategy, the "processing time per request [scales] linearly with the logarithm of group size." Unfortunately, the "signing [of] rekey messages increases the server processing time by an order of magnitude."

The high communication and processing costs incurred in key management, and the implicit trust of the cloud provider required, cast doubt on a classic centralized solution as a viable candidate for a mobile cloud computing system.

3.1.2 Identity-Based Encryption

One general problem with PKE that needs to be addressed is that a mechanism is needed for users to find and obtain the public keys required for encryption, whether keys correspond to individual users, to a data partition accessed by a set of users, or to individual records. To simplify certificate management, *Identity-Based Encryption* (IBE) was invented, based on BDH (Bilinear Diffie-Hellman) [9, 19]. The public key used in this scheme is derived from an arbitrary string such as an e-mail address that can uniquely identify a party; it dispenses with the need to query a key authority; this concept is used to reduce the communication overhead of requesting encryption keys from the cloud provider, and has the added benefit of enabling multi-user access to shared data. Thus, querying a certificate

authority for public encryption keys on demand, as with RSA, is not required, reducing the cost of communication. An expensive pre-distribution of authenticated keys is also unnecessary, unlike in a traditional public-key infrastructure. It has been demonstrated that an IBE technique can be faster than one based on RSA. One protocol called Identity-Based Hierarchical Model for Cloud Computing (IBHMCC) has been proposed with faster authentication and less communication cost than the SSL Authentication Protocol (SAP) used on many web servers [71]; it provides encryption by generating and storing secret keys at two different levels: the cloud data centre and the users. The encryption key is derived from the public keys of the data centre, the originating user, and the target user; unfortunately, this method restricts data exchange to occurring between two individual parties without consideration of a multi-user setting. Although IBE generally helps reduce the communication cost entailed with key distribution by allowing users to generate keys themselves, it is lacking in that it still requires trust in the cloud provider.

To address the issue of trust, Certificateless Public Key Cryptography (CLPKC) may be utilized [3]. A Key Generation Centre (KGC) resides in the cloud but "[it] does not have access to users' private keys." The KGC supplies each user with a partial private key PSK which the KGC computes from an identifier for the entity and a master key. The user then combines his or her PSK with some secret information to generate a full private key SK. In this way, the user's final private key is not made available to the KGC. The user also combines his or her secret information with the KGC's public parameters to compute his public key PK. In fact, the user need not be in possession of SK before generating PK: "all that is needed to generate both is the same secret information." The advantage of this scheme is that it prevents the KGC from being able to decrypt all data and communications directly. One problem, however, is that "the system is [no longer strictly] identity-based, because the public key is no longer computable from an [identifier]." Also, "the KGC needs to ensure that the partial private keys are delivered securely" to users using some available secure or out-of-band transport.

3.1.3 Hierarchical Access Control

As described in the use cases for mobile cloud computing, it is useful to control permissions for users at different levels within an organization. Traditionally, the management of keys to provide hierarchical access control has assumed the presence of a trusted Central Authority (CA) that maintains the keys. Many such solutions rely on a tree-based structure of nodes, where any node can derive keys for its descendants. Major challenges include the rekeying operations that must occur whenever nodes are added or deleted in the tree. One scheme strives for efficiency by utilizing less expensive hash operations to avoid expensive re-keying and allow for efficient key derivation for nodes that are lower in the access hierarchy [7]. In another scheme, for the purpose of securing access to clouds, it is suggested that identity-based encryption be combined with a hierarchical access control scheme, such that every Private Key Generator (PKG) maintains the private keys of all users in its own domain [118]. A root PKG generates private keys for lower-level PKGs. The presence of multiple PKGs in a cloud system suggests an inefficiency, however.

3.1.4 Distributed Key Management

To transfer key management duties and associated trust from the cloud provider to users, a distributed key model may be used in place of a centralized one. Shares of a private key are distributed among a number of trusted users. Any authorized member of a group can request sufficient shares of the private key associated with a single identity and reconstruct the entire private key; this objective may be accomplished using the concept of *partial keys*. On the basis of IBE, it has been "suggested that distributed PKGs [can] function as decryption servers [to implement] *threshold decryption*;" [10] each PKG holds a share of the master key, and a sufficient number of these shares must be assembled by a user [19]. This arrangement requires each PKG to be involved in communication with users at all times, because the key share needs to be re-distributed for each new ciphertext; this is inefficient in a large user set. It has been proposed that a private key received from a PKG, rather than the master key, can be distributed among several users in portions such that a sufficient number of portions are required to decrypt a message, thus removing the requirement for central key storage and constant involvement from the PKGs [10]. Any authorized member of a group can request sufficient portions of the private key associated with a single identity, reconstruct the entire private key, and distribute it within its domain.

The problem remains that a single mediator is still responsible for assembling a key from multiple sources, such as decryption servers, and distributing it for each decryption; the danger is that this can result in a bottleneck in a cloud system; furthermore, key revocation is expensive.

The threshold decryption approach may be refined to manage sessions. By using IBE, it is possible for a public key to be constructed using the recipient's public identity joined with a timestamp; the encrypted message can only be decoded by obtaining the private key from the PKG generated using that timestamp [19]. Once access is revoked, the PKG stops generating new private keys; although this effectively enforces a time limit for keys and limits the danger of compromised keys, the constant key updates imply that this approach is impractical for cloud applications, however.

A comparison of the key management approaches discussed appears in Table 3.1.

Responsibility for storage of key material	**Advantages**	**Disadvantages**
Centralized in the cloud provider.	Utilizes the scalable computational and network resources of the cloud. Relies upon the direct user-to-cloud link.	Requires trust in the cloud provider to not decode encrypted user data stored on its servers.
Centralized in a trusted authority that is outside of the cloud domain.	Does not require trust in the cloud provider. May control access to cloud data through an intermediary node.	Requires maintenance of a scalable authority server by the client, or trust in a third-party guardian as a paid service.
Fully decentralized among users.	Requires no additional network elements. Key sharing may utilize cheap local links such as Wi-Fi or Bluetooth.	Obtaining keys may require arbitration by an authority which entails additional traffic. Revocation is inefficient.

Table 3.1: A comparison of approaches to key management in mobile cloud computing.

3.1.5 Proxy Re-Encryption

The shortcomings of traditional key management are increasingly being addressed by recent ideas in secure cloud storage. The goal is for data stored in the cloud to be encrypted to maintain its confidentiality from the provider. One option is to utilize a trusted authority administered by the client to retain secret key material; the challenge is that it must be sufficiently scalable to perform key distribution duties for a large population, which is expensive to engineer. Also, an inherent risk of the authority is that it constitutes a single point of failure. The main difficulty, however, arises when new mobile users join the system, and existing ones leave, at a great frequency; this necessitates re-generating keys and making cloud data inaccessible to those who have left. Having an authority fetch data from the cloud, decrypt it, and then re-encrypt it with a new key is unrealistic from a performance standpoint, especially in a highly scaleable system.

The dilemma is that the cloud provider has nearly unbounded computational resources to perform cryptographic operations, but it cannot be trusted with key material, while an external trusted agent suffers from scalability potential. One solution entails cloud data being transformed such that it may be unlocked only with newly generated keys, without needing an intermediate decryption step; this is possible by having the cloud provider perform a re-encryption operation without having access to the actual decryption keys. An active area of research, it is a form of *proxy cryptography*, where a third-party such as the cloud re-encrypts content for a user. One possibility is to utilize unique content decryption keys that are further locked so that they cannot be accessed by a cloud provider [8]. Users download encrypted user content, then request that a trusted access control server provide them with the appropriate content decryption keys, by performing re-encryption. In such schemes, the content owner typically decides which users should have access to cloud data and allows re-encryption to be carried out accordingly. If a user's access rights are revoked, then no re-encryption will occur for that user.

The challenge in a mobile cloud computing context is that the content owner, which is a mobile device, manages access control for all other users; it can become an excessive communications burden if it requires constant activity by the data owner such as generation and distribution of new versions of keys. Also, the appropriate timing of re-encryption tasks, which are very computationally-intensive, remains an open problem; in some cases, they may be performed only when needed, or at cheaper off-peak times, resulting in *lazy revocation*.

Another area of research that is correlated is *attribute-based encryption* [50], where a recipient may be able to read an encrypted message based on the satisfaction of certain attributes, rather than possession of a valid key, at the cost of additional complexity; examples include membership in a particular department or authority level within the organization. Regardless of the re-encryption mechanism used, data stored in the cloud should remain encrypted in some form at all times, and any required transformation of it should not reveal the plaintext in the process; such an agenda is in keeping with the threat model identifying the cloud provider as being honest-but-curious.

Re-Encryption System Model

A system model for proxy re-encryption is now presented, which is a variant of the general mobile cloud computing system envisioned in Section 2.2.2 on page 14. Figure 3.1 illustrates a typical proxy re-encryption scenario. The *manager* is a trusted self-supporting network component that may be situated behind an organization's firewall and form part of a private cloud belonging to the client. It is normally administered under the domain of the users in question and is completely independent of the CSP. It may maintain a database of private key information relating to a set of authorized mobile users. The manager is sufficiently trusted to authorize access to the cloud and to contain key material as necessary; however, to minimize the risk of it being compromised, a user will only share as much of its own key material with the manager as is necessary in the security scheme utilized. Furthermore, the manager will not be as inherently scalable as a cloud provider.

Inside the cloud, the controller maintains a complementary public key information database, and stores and reads user data on behalf of clients to and from the permanent replicated data store. The user data may periodically undergo cryptographic transformation, such as re-encryption from one version of ciphertext to another; such activities are dispatched on-the-fly or alternatively at off-peak times by eligible worker processes instantiated and initiated by the controller.

A mobile user may act as a *data owner* and decide what access privileges are appropriate for the data that it uploads to the cloud and retains control over; a specific subset of the user population may be identified as having sufficient permission based on unique identities, or users may be assigned various distinguishing attributes that inherently grant permission regardless of the specific identity that assumes them. A highly scalable system

is envisioned where users may potentially number in the many thousands or millions. Continuous arbitration by a single data owner during all transactions is impractical, as a mobile user is subject to a limited battery and transient connectivity. The data owner uploads encrypted content to the cloud data store using a shared public key; another user requests it, and the manager invokes a re-encryption process either within the cloud or inside itself; the content is then downloaded directly from the cloud or via the manager, and read by the recipient using his or her own private key. If the recipient's access rights are revoked, then content will not be re-encrypted to a readable form for him or her.

3.1.6 Encrypted File Storage

Various network file systems have been proposed to provide encrypted file storage capability for users. Although these systems were not specifically designed for use in cloud computing, it is instructive to examine the approaches taken to solve the problem of scalable security.

An oft-cited system is SiRiUS, which essentially adds a security mechanism on top of an existing networked file system [58]. It supports granting read-only or read-and-write access to files by providing file encryption and signature keys that can access a data block; these are stored as metadata along with the data block, and are encrypted using the data owner's master encryption and signature keys. When sharing, an encrypted data block is created consisting of these access keys encrypted using the intended reader's public key. Unauthorized modifications to files, such as attempts to reverse an occurrence of revocation, are detected by storing a hash tree of the metadata, and comparing it against newly modified metadata by users. SiRiUS supports only end-to-end security; all cryptographic operations must be performed on the client, which may be unrealistic in the case of a mobile device. The system relies upon an existing key distribution system using a Public Key Infrastructure (PKI) or IBE. In particular, SiRiUS employs an encryption key permitting read access to a data block, and a signature key permitting write access. Scalability is a major issue with this system. A user wishing to share data with another user must encrypt the file encryption key using the public key of the target user; repeating this process for a large target population of users, some of which may be unknown at the time of data creation, is problematic. Key management is simple, however, as a user only needs to retain a master key pair, and out-of-band communication is not needed if an IBE scheme is utilized for public key retrieval. To address the problem of allowing a large number of users to access the same data block in SiRiUS, the Naor-Naor-Lotspeich (NNL) subset-sum

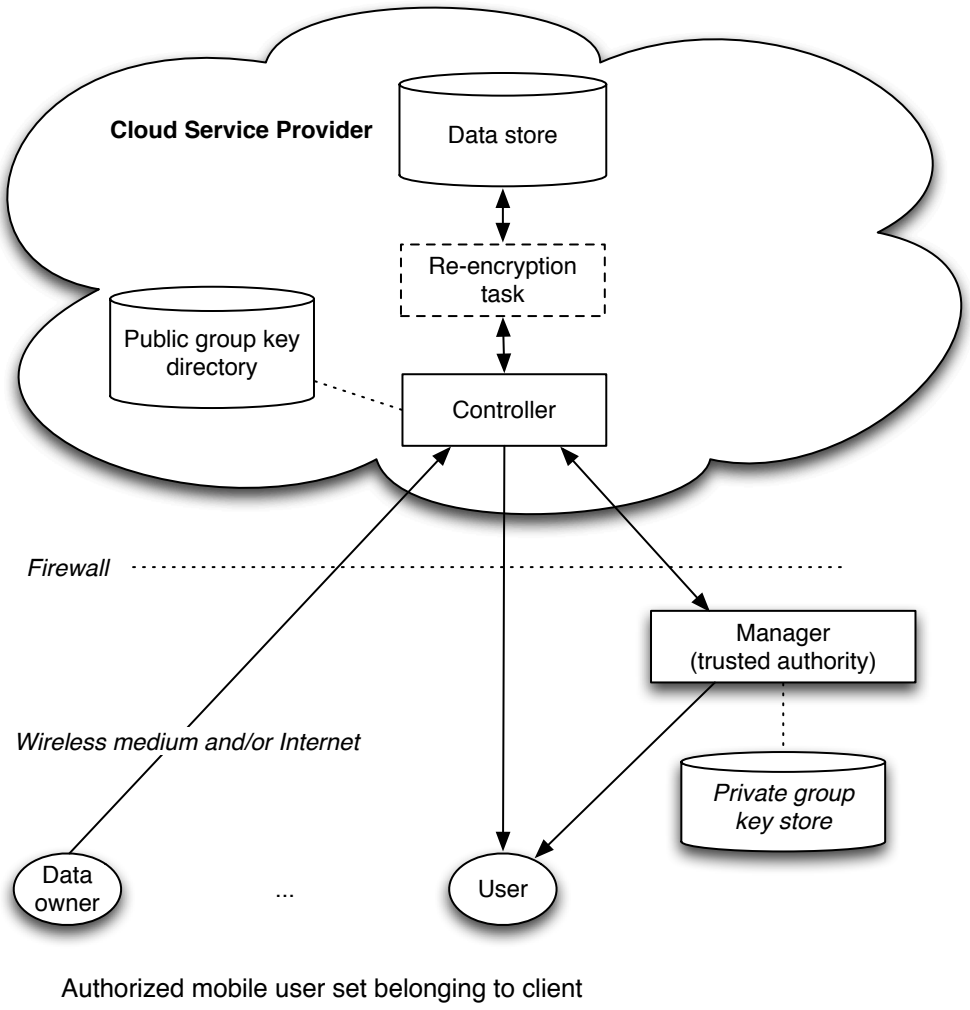


Figure 3.1: A system model of proxy-based data re-encryption.

framework may be utilized. Here, the potential recipients are grouped into subsets, and the encryption key is encrypted using the key belonging to a whole user subset, and not just an individual user. To reduce the cost of revocation, "each subset is a complete subtree rooted at some node in a tree. A user is given keys corresponding to those subtrees rooted at nodes along the path from [his or] her leaf to the root." To support this scheme, the expanded key blocks in the file metadata require significantly more data storage, however, which is a tradeoff in achieving scalability. Ultimately, this cryptographic file system relies on the client performing all cryptographic operations, which is a significant disadvantage.

Plutus is another secure file storage system with *encrypt-on-disk* capability that offloads all key management and distribution to the client [60]. The motivation behind its design is that the cost of encryption and decryption is distributed among all users without server involvement. This approach is not very feasible in a cloud-based system consisting of clients with resource-constrained devices, however, where all wireless communication and processor-intensive cryptographic operations should be kept to a minimum, and a secure out-of-band channel is generally unavailable. One of the defining features of Plutus is that it offers *lazy revocation*. Following a revocation, a revoked reader may still read unmodified or cached files; it cannot, however, read updated files or modify them itself. A major limitation is that users must exchange keys securely between themselves; no such mechanism to accomplish this feat is proposed. In addition, readers must contact the file owner to obtain the necessary keys for decryption, which further increases the communication cost. Scalability in Plutus is achieved through *filegroups*, which are essentially aggregations of files shared by multiple users that have a single RSA key pair associated with them. The vulnerability of the file system is reduced through the use of *lockboxes*. Lockboxes contain the keys for individual files, and the lockboxes are themselves encrypted with an RSA key. Enforcement of read versus write access is achieved through cryptographic means, as opposed to relying upon the server. Writers receive file-sign keys while readers receive file-verify keys. Readers verify the integrity of data downloaded from the server by checking the signature of each block. To address the problem of an increase in the number of keys due to revocation, a technique called *key rotation* is utilized. Files are re-encrypted with the latest keys, and users can derive earlier keys if needed, as all keys are related to each other. However, only the owner can generate the next version of the key; this prevents a revocation from being undone by a reinstatement of an older key, but it requires availability of the original owner of a file. In a large-scale multi-user cloud application, the required availability of the data owner may be infeasible. Indeed, necessarily identifying a single

user as a permanent owner is also unrealistic. Key rotation is done by exponentiating the current key with the owner's private key, which is a relatively expensive operation. Another feature of Plutus is that the server can validate writers of files based on their supply of a write token issued by the file owner; the hash of this write token is stored on the server for verification. This is not a significant improvement over a traditional access control list securely maintained on the server, however, and the issuance of a large number of write tokens by a mobile user is unrealistic.

Tahoe is a storage grid designed to provide secure, long-term distributed storage [115]. It is especially useful for backup applications and can be used in a RAID-like manner. It is designed to remain operable even if some servers across which a file is shared become unavailable. The motivation behind Tahoe is the *Principle of Least Authority*, where cryptographic capabilities grant the minimal set of privileges necessary to accomplish a task. The cryptography employed by Tahoe ensures that servers cannot violate confidentiality by reading the original plaintext, nor can they violate integrity by forging file contents. It utilizes a *capabilities-as-keys* access control scheme, which enables a client to gain access to an object based on knowledge of an identifier. Capabilities for reading from, writing to, and verifying the integrity of files are provided. The client chooses a symmetric encryption key, encrypts the file, creates multiple shares encoded with erasure codes for integrity verification and file reconstruction, and then uploads the shares to different servers. This approach does not fit with a cloud computing system, where the client expects a single upload to a cloud provider to occur, and has no knowledge about multiple servers contained within the cloud; the cloud provider itself is responsible for replication if necessary. Tahoe requires the use of public-key encryption such as RSA for key generation, but does not provide a facility for key distribution between users, nor a mechanism for revocation. Freshness of mutable files is handled by having the client request metadata about a data share from a sufficient number of servers, identify the highest-numbered version, and confirm that enough shares of that version exist. Again, this solution is not a close fit to mobile cloud computing; having a mobile user communicate with multiple servers is infeasible.

CryptDB [93] is a system that allows execution of SQL queries over encrypted data; it also ensures that curious database administrators do not gain access to encrypted data. "Each data item in the database can be decrypted only through a chain of keys rooted in the password of one of the users with access to that data. As a result, if the user is not logged into the application, and the adversary does not know the user's password, the adversary cannot decrypt the user's data." The main difficulty with this approach is that

CryptDB assigns and manages keys for principals (physical users) itself; keys for use in a session are obtained from a user that is logged-in, and the data belonging to that user can be compromised while accessed. The trust model is relaxed; the server must be trusted to restrict permissions for the administrator, and to delete the user's key on log-out; such compromises are made to permit queries to be efficiently executed on unencrypted data. Furthermore, the server performs no re-encryption function itself when a user is off-line.

Clearly, the designs of encrypted file storage systems are rooted in a traditional client-server setting that does not impose the steep scalability and efficiency requirements of a mobile cloud system. Cryptographic operations and key distribution tasks are typically outsourced to the clients, rather than the network storage system, which is opposite to what is desired in a cloud accessed by resource-limited mobile devices. High availability of data owners is also required, but mobile devices suffer from transient connectivity and provision of such an assurance is unrealistic. Furthermore, encrypted file storage systems may be implemented on remote machines that do not guarantee a particular level of service, whereas a cloud system is a tightly-controlled collection of nodes under single ownership that exists on the basis of delivering a service under contract with high guarantees.

3.1.7 Secure Cloud Storage

Recently, secure storage system design has migrated from the network file system domain to that of cloud computing with more tailored solutions, yet work in this area is still nascent. CloudProof is a secure storage system designed for the cloud [92]; "[clients of this system] can not only detect violations of integrity, write-serializability, and freshness, [but] they can also prove the occurrence of these violations to a third-party." The system does not deal with confidentiality, however. It presents an auditing protocol that uses *attestations*, or proofs, "which are signed messages that bind the clients to the requests they make and the cloud to a certain state of the data;" this allows violations to be proved. CloudProof manages access control by grouping data blocks into a family, such that each block in the family is managed by a single ACL that dictates what group of users should have access to the block family. All of the key information is stored in the family key block. The read access key pertaining to a block family is distributed to all users using broadcast encryption. The main challenge lies with revocation of user access. Immediate revocation would entail simply re-encrypting all blocks managed by ACLs that include the affected group, which would be expensive. Instead, CloudProof performs *lazy revocation*. Using

key rotation as in Plutus, the data owner rolls keys forward to a new version for each of the affected families. The new key information is sent to all current authorized users via broadcast encryption, which has complexity equal to the root of the membership in the ACL. Upon data access, a client checks the version of the read access key; if it is out of date, then the client re-encrypts the data. Thus, the revocation burden is borne by users; a data write always requires re-encryption, anyway. This scheme seems unworkable in the context of mobile users: they cannot be expected to perform re-encryption operations themselves given their limited computational capacity. The cloud provider does not assist with the expensive re-encryption activity, and the data owner is responsible for performing the key rotation. In practice, a mobile user would not be expected to be available and connected at all times to perform such activity, nor could it endure the communication cost of it.

DepSky is a cloud storage system that addresses secure storage on multiple clouds; it permits "encryption, encoding, and replication of data on diverse clouds that form a *cloud-of-clouds*" [16]. "All writers of a data unit share a common private key used to sign some of the data written on the data unit, while readers have access to the corresponding public key to verify these signatures." It is assumed that the system has some sort of access control, but it is left unspecified. Confidentiality of data stored in the clouds is ensured without the use of a key distribution service, by employing Shamir's *secret sharing scheme* [98], where a "dealer distributes a secret to [all] players, but each player gets only a share of the secret;" a sufficient number of shares are needed to recover the secret key. In the case of the basic variation of DepSky, the players are the participating clouds, not the users. An erasure code algorithm is employed that reduces the size of each share, such that "data is encrypted with a random secret key, [it is then] encoded, [and] the key is divided using secret sharing; each server receives a block of the [same] encrypted data and a share of the key." "No individual cloud has access to the data stored, but clients that have authorization to access the data" must communicate with a sufficient share of different other clouds to obtain all key components and rebuild the original data. This communication cost is excessive for a mobile user, and it also requires full data replication on all clouds. There is also the practical difficulty of finding sufficient non-collaborating cloud providers to implement the proposed system. In another variation of DepSky, to reduce cloud storage demands, the data itself is partitioned into a set of blocks such that a sufficient share of blocks is necessary to recover the original data. Each reader requires multiple replies to a read request, and a data writer is required to generate and store a message digest for each cloud; these features also entail a high communication cost.

The CS2 cloud storage system [61] provides facilities to conduct symmetric search of large volumes of encrypted data with provision of data confidentiality and global integrity; however, it does not address issues of key management and end-to-end security as it considers itself to be of a different purpose than a cryptographic file system for the cloud.

The secure cloud storage systems detailed here impose substantial communication and processing costs on users, such as on-the-fly data re-encryption or constant communication with multiple clouds, as described. These demands would prove insurmountable for resource-constrained mobile device users. Clearly, new solutions must be designed such that mobile users are accommodated without compromising data privacy and integrity.

3.2 Security Features in Commercial Clouds

Inroads have been made in standardizing key management for commercial cloud use. OASIS (Organization for the Advancement of Structured Information Standards) has proposed KMIP (Key Management Interoperability Protocol) for unified cloud management [87], which addresses interoperability of key management services in use in the industry; it defines a single low-level standardization protocol for communication between key management systems and applications within an enterprise system, but it arguably falls short of addressing the unique scalability problems inherent in cloud computing. It has been observed that KMIP has a limited focus on enterprise key management and lacks applicability towards cloud computing; there is agreement over the need for a scalable key management model applicable to today's cloud systems [77].

Authentication is an important aspect closely related to key management; it determines whether a user can gain entry to a cloud. This function may be accomplished through the use of SAML (Security Assertion Markup Language), permitting an organization to manage authentication for its own users, as well as between other sites using trust relationships. It provides Single Sign-On (SSO) capability, which is seamless authentication on multiple cloud providers; a user may log into multiple sites using a common identity authorized by an identity provider. Microsoft's Azure cloud computing platform [24] provides access control by having each user supply a token that contains claims, which is more efficient than individual access control. An identity federation scheme permits claims created in one identity scope to be accepted in another. Amazon's EC2 (Elastic Compute Cloud) [5]

requires multi-factor authentication; customers must not only supply their standard credentials, but also supply a single-use code from an authentication device in possession.

Network security on a commercial cloud systems is largely handled by a firewall that protects a running virtual instance from attack by allowing only certain ports to be accessed from outside by legitimate clients. In EC2, a mandatory inbound firewall is configured in *deny mode* by default; customers must explicitly open all ports to allow inbound traffic. Communication may be restricted by protocol, service port, and source IP address.

Other security mechanisms relate to the inner workings of the cloud. One example is *process isolation*, where a sandbox provides a controlled set of resources for programs to run in, while restricting communication access; it permits untrusted programs to co-exist safely in the cloud. In the secure sandbox environment of Google's App Engine (GAE) platform [49], each cloud application runs in a separate process, and the data store prevents an app from accessing data belonging to other applications. A cloud application is forced to access external computers solely through a provided URL fetch service, and inbound traffic occurs on standard HTTP ports. An application is prevented from writing to the file system directly, and must return a response to a web request within a specified time.

In the case of an untrusted cloud provider, key management services are sometimes handled by a third-party guardian. For example, the security provider enStratus enforces a separation of roles [39]; it acts as a custodian of the client's security keys and credentials. It maintains the key store outside of the cloud but has no access to client data, while the cloud provider is the opposite: it holds the encrypted data, but not the keys. The security of this approach lies in the difficulty of successfully attacking two independent and encrypted systems. However, the client credentials are stored on one of enStratus's own servers, and although it is allegedly not externally accessible, it could still be compromised; clients may also be unwilling to trust a third-party provider with their keys any more so than the CSP.

Generally, commercial cloud providers are innovating security features, but solutions tend to be fragmented and proprietary, and tailored for specific types of applications; in particular, for web server traffic. Evidently, there is no readily available and efficient key distribution scheme to provide fine-grained access control to encrypted storage in the cloud.

3.3 Summary

The lack of direct control over the resources of a cloud computing system require clients to either fully trust the cloud provider, or adopt cryptographic protocols to ensure the security of data that is transferred to and stored in the cloud. The increasing popularity of cloud applications that service mobile users implies that these protocols must be made efficient. To provide trusted, encrypted communication with a cloud application, a highly scalable key management system is needed. Data confidentiality must be preserved through in-transit data encryption as well as encrypted data storage. One important feature in particular is efficient key revocation, due to the frequent changes in membership that can occur in a large multi-user cloud application. Furthermore, key management solutions in use by commercial cloud systems today are largely insufficient from a scalability point of view. Security technologies adopted by general-purpose web servers and networked file systems also do not fully address these needs. In the following chapters, a number of novel key management techniques to achieve scalable and secure mobile cloud computing will be proposed and analyzed.

Chapter 4

Re-Encryption-Based Key Management

4.1 Introduction

KEY MANAGEMENT TECHNIQUES are proposed in this chapter which minimize communication sessions for users and avoid expensive client-side calculations; they take advantage of an asymmetric computing model in which the cloud server has much greater computational ability than a mobile client to process cryptographic functions of a security protocol. The re-encryption system model described in Section 3.1.5 on page 37 and the adversary and threat model described in Section 2.4.1 on page 19 are assumed.

First, related work is presented on proxy re-encryption in Section 4.2, beyond what is described in Chapter 3 on page 30. Two key management schemes based on data re-encryption are then presented: a conventional one entailing a re-encryption workload processed by a trusted client-controlled party, with novel improvements suggested, in Section 4.3; and the other entailing a workload processed by the cloud provider as an untrusted entity that offers greater scalability in the mobile cloud computing context, in Section 4.4. Implementation results on popular smartphone and cloud platforms are provided that validate the assumptions made, in Section 4.5. Finally, concluding remarks on the efficiency and scalability of the proposed schemes are made in Section 4.6.

The content of this chapter is based on work that has been published [108, 109].

4.2 Related Work on Proxy Re-Encryption

Data stored in the cloud should ideally be stored in encrypted form so that the cloud provider cannot access it. This notion is dependent on the keys being securely managed by an entity outside of the provider's domain. The difficulty arises when new users join the system, and existing ones leave, necessitating new keys to be generated. The encrypted data should ideally be transformed such that it may be unlocked with new keys, without an intermediate decryption step that would allow the cloud provider to read the plaintext.

One possible method is to re-encrypt the stored content during retrieval. Such a technique has been applied to an encrypted file storage system using proxy cryptography and the Chefs file system, where a "[content] owner encrypts blocks of content with unique, symmetric content keys [(symmetric keys presumably being preferred for performance reasons); these keys] are then encrypted with an asymmetric master key to form a *lockbox*" [8]. "Users download the encrypted content from the block store, then communicate with an access control server to decrypt the lockboxes protecting the content. [Critically,] the content owner selects which users should have access to the content and gives the appropriate delegation rights to the access control server." To accomplish this, the content owner retains a master key that is used to compute a re-encryption key; this re-encryption key is used by the "access control server to re-encrypt the lockbox [to that of the intended reader's] public key." The problem with this approach is that the content owner manages access control for all other users, which is a great burden on communications if the owner is a mobile device user. In addition, it requires dynamic re-encryption of the same data whenever multiple users want to access it. In the novel model proposed later in this thesis, one-time re-encryption only occurs whenever membership changes, presumably a less frequent occurrence than that of data access. Also, access rights need not be enforced by individual users, and it is not possible for a single user to divulge the keys of all other users to the cloud provider, as they are not known. Other approaches exist [57] that also require a trusted proxy for each decryption, which increases the communication cost.

A related work proposes the merging of Attribute-Based Encryption (ABE) with proxy re-encryption in a cloud computing application, allowing fine-grained access control of resources while attempting to offload re-encryption activity to the cloud provider [119]. This scheme has numerous differences to the cloud-based re-encryption scheme that will be proposed; these differences prove to be disadvantageous in a mobile-based environment. The data owner, or originator, is involved in generating a key for each new user that joins

or leaves the system, rather than offloading this task to a trusted key authority under the client's control. This is not only a prohibitive cost for a mobile user, but also impractical due to the user's mobility and hence occasional unavailability. Another difference is that a secret key must be regenerated and re-distributed for each user, in lazy fashion, whenever user revocation occurs, rather than allowing users to upgrade a common partition key based on public parameters which would reduce communication and result in higher efficiency. Also, the data re-encryption activity is aggregated in lazy fashion, whereas in this proposal, re-encryption occurs dynamically on an as-needed basis, greatly reducing server workload for data primarily accessed by approximately the same set of users over time. The re-encryption occurs due to attribute re-definition, unlike the proposal. There is also no facility for exchanging key material in peer-to-peer fashion, which would be useful among mobile users utilizing cheap local wireless links such as Bluetooth. Finally, the scheme is based on KP-ABE (Key-Policy Attribute-Based Encryption), not CP-ABE.

Similar observations are made with respect to another related approach that combines Hierarchical Identity-Based Encryption (HIBE) and Ciphertext-Policy Attribute-Based Encryption (CP-ABE), which uses hierarchical domain masters to distribute user keys and the cloud provider to re-encrypt data on user revocation depending on the attribute keys held by the revoked user [113]; this is done at the cost of increased storage requirements for key material held by users and a greater amount of processing when generating ciphertext, which are problematic for mobile device users. Another method of trusted data sharing over untrusted cloud providers has been proposed that uses a progressive elliptic curve encryption scheme [123]. However, it relies upon a writer uploading encrypted data to the cloud, then distributing credentials to the cloud to perform re-encryption, and also to the reader on each data access attempt; this is clearly impractical when applied to resource-constrained devices and networks. The inefficiency of peer-to-peer key distribution in this manner is best avoided.

Although data re-encryption appears to be a promising technique in managing encrypted data as access rights evolve over time, current solutions in the literature do not address the issue of high scalability to a sufficient and satisfactory degree.

4.3 Manager-Based Re-Encryption

4.3.1 Introduction

The overall goal of this chapter is to explore, adapt, and evaluate system security engineering techniques to achieve a high level of communication security for cloud computing systems. In particular, the emphasis will be on the scenario of a mass multi-user services application running in the cloud and interacting with a high population of active mobile device users. A key management scheme is described in this section that is based on the proxy re-encryption cryptography suggested in [8]; however, it has been mapped to a cloud computing system with significant modifications. Its primary involvement here is to demonstrate a technique that will serve as a foundation and point of comparison for the novel scheme proposed in the following Section 4.4 that offers much greater scalability. Some additional novel variations are still suggested at the end of this section, however, which provide limited but substantive improvements to performance, scalability, and security. The contributions of this chapter include the adaptation of proxy cryptography to a cloud computing system such that the communication cost for mobile users is reduced, as compared to the encrypted file system application in [8] that is less practical in the mobile usage context studied here, as will be explained.

The scheme described in this section permits access to a common data partition in the cloud among multiple users, ensures confidential data storage to which even the cloud provider is not privy, and offers greater data access efficiency in a mobile-based cloud system at lower overall communication and processing cost than traditional centralized solutions; all of these features are accomplished through the process of data re-encryption. Table 4.1 summarizes the notation used throughout.

A manager, or trusted proxy node, controls the access of its users to the cloud. This manager is typically under the control of the client organization, and ensures that key management functions need not be outsourced to an untrusted cloud provider. The manager may comprise a server situated behind the firewall of the client organization that is securely accessed by a mobile user population. At the same time, the cloud stores user data in encrypted form such that it is accessible to all authorized users at any time; it does so by regularly performing one-way re-encryption of the data in the cloud as it is being accessed, so that a reader in the authorized group can decode it using the reader's own decryption key.

Symbol	Description
P	Cloud data partition.
$\mathcal{P}$	Set of all partitions.
U_P	User group with authorized access to P.
M	Manager or trusted proxy.
A, B, C	Users Alice, Bob, Charlie.
m	Plaintext message.
$E_x(m)$	Ciphertext of message m encrypted using secret key x.
PK_{X_v}	Public key of entity X (with version v optionally specified).
SK_{X_v}	Decryption key of entity X (with version v optionally specified).
$RK_{X \to Y}$	Re-encryption key for converting from content unlocked by SK_X to that unlocked by SK_Y.

Table 4.1: A legend for the symbolic notation used in the re-encryption models.

4.3.2 System Operation

Key Generation and Encryption

Consider modifications and improvements to a proxy re-encryption scheme [8], based on the BBS encryption method [18] and the El Gamal crypto-system [38]. The proof of the underlying encryption technique is presented in [8], and is relied upon here.

As shown in Figure 4.1, the manager generates public and private decryption keys (PK_X and SK_X) for each user X belonging to the system, and is responsible for maintaining an access control list for enforcing the authorized user set. A data partition P in the cloud is accessible by a user group U_P and belongs to the entire set of partitions $\mathcal{P}$. In this example, Manager M manages the access of user group U_P to data partition P. Note that a single user may belong to multiple groups.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be groups of prime order q with a bilinear map such that: $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. The system parameters are the random generator $g \in \mathbb{G}_1$ and $Z = e(g, g) \in \mathbb{G}_2$. A secret decryption key SK_X is randomly selected by M for each user $X \in U_P$ and distributed to the users through a secure (possibly out-of-band) channel. Let: $SK_X = x \in \mathbb{Z}_q^*$. A public key PK_X is also chosen for user X as follows: $PK_X = g^x$. Similarly, the manager M also creates a private key $SK_P = p \in \mathbb{Z}_q^*$ and public key $PK_P = g^p$ for data partition P in the cloud that it manages. The public partition key may reside in a directory inside the cloud

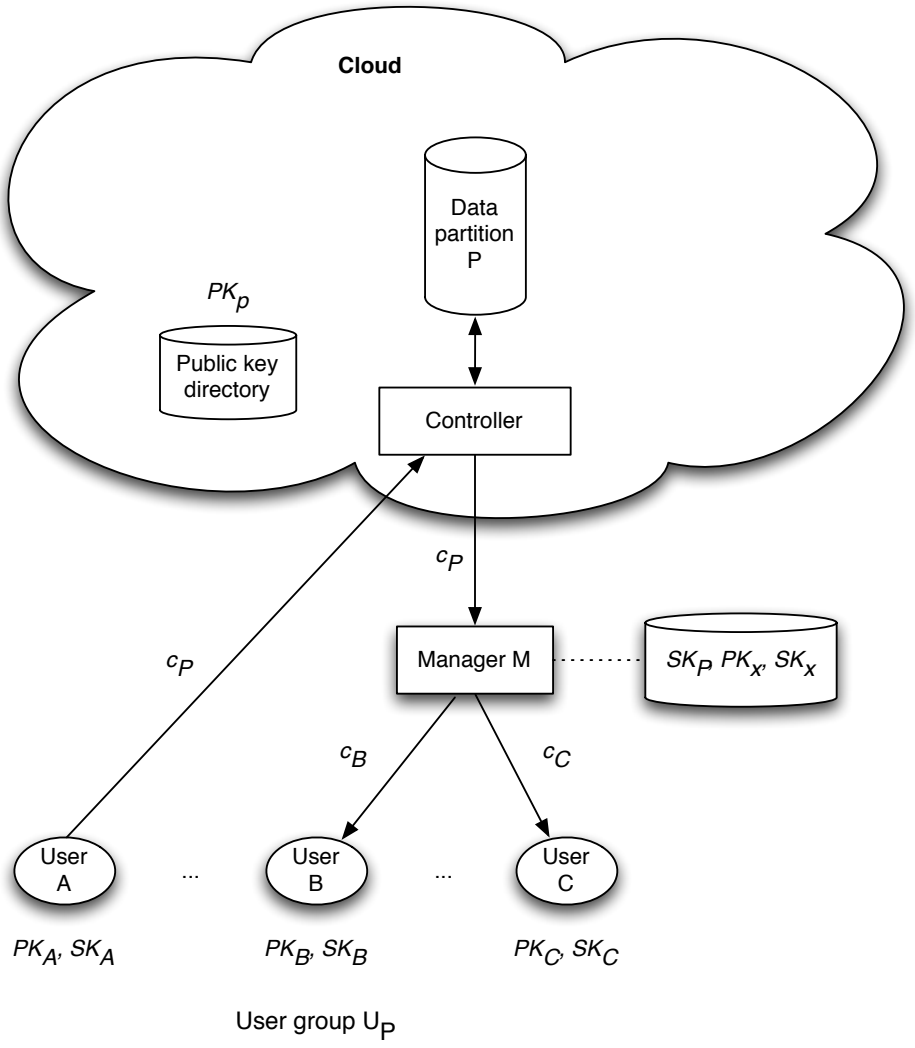


Figure 4.1: A model of key management using manager-based re-encryption.

that is accessible by all users in the system, or be distributed to all users in U_P by the manager; it is considered public information. The manager, however, retains the private decryption key SK_P required to read the cloud data; the cloud provider and other users cannot decode the data even if they download it directly from the cloud, with or without authentication. A unique property of this model is that all read requests initiated by users are normally serviced through the manager.

User A, or Alice, encrypts a message m and creates a ciphertext using the public key PK_P of the data partition where it is to be stored, and uploads the cipher-text $E_p(m)$ to the cloud, so that it is stored in encrypted form in partition P.

$$\text{Given } m \in \mathbb{G}_2, \text{ random } r \in \mathbb{Z}_q^*$$

$$E_p(m) = (Z^r \cdot m, g^{pr})$$

The cloud provider will be unable to extract the original content m. If m exceeds the maximum possible block length, then the message may be segmented, and each segment encrypted using the same key PK_P. Alternatively, a symmetric cipher such as AES-256 may be applied to the entire message, and the cipher key itself encrypted using the proposed scheme, as opposed to the entire message. Irrespective of the approach taken, the number of required encryption keys in the proposed scheme will not increase, as the same pre-computed public partition key is applied in all encryptions relating to the same message.

Re-Encryption

Suppose that a user B, or Bob, belonging to the same group, makes a request to the cloud provider for the same message m stored earlier by Alice. The cloud provider does not send it to B directly; instead, it sends it to M, which decides whether that data should be accessible by B based on its ACL. If so, then the manager creates a re-encryption key $RK_{P \to B}$ using the private key of the partition. The manager then fetches the encrypted message $E_p(m)$ from the cloud, and computes a re-encryption key using B's decryption key SK_B, which was initially generated by the manager and shared with B. Note that SK_B is equal to $b \in \mathbb{Z}_q^*$, chosen randomly by M. In general, the re-encryption key computed for user X in U_P is:

$$RK_{P \to X} = g^{\frac{SK_X}{SK_P}} = g^{x \cdot p^{-1} (\text{mod } q)}$$

For user B, as in this example, the re-encryption key computed is $RK_{P\to B} = g^{\frac{b}{p}}$. Using this key, M re-encrypts the ciphertext $E_p(m)$ as $E_b(m)$ and sends it to B directly.

$$\begin{aligned} \text{From } E_p(m) &= (Z^r \cdot m, g^{pr}), \\ \text{Compute: } e(g^{pr}, RK_{P\to B}) &= e(g^{pr}, g^{\frac{b}{p}}) = Z^{br} \\ \text{Publish: } E_b(m) &= (Z^r \cdot m, Z^{br}) \end{aligned}$$

The justification for re-encrypting data before delivery to B is as follows:

1. A secure channel between M and B may not exist, and hence encryption is required.
2. M may perform a re-encryption without possession of the private key of B, and is unable to read the ciphertext during the process; this possibility is explored in a variation presented later in Section 4.3.4, and is not possible if M was to follow a two-step process and simply decrypt then encrypt the data with the recipient's private key. Thus, the security of the proposed scheme may be enhanced with re-encryption, at the cost of a simpler and faster scheme using symmetric keys.

Decryption

The recipient B can then decode the ciphertext $E_b(m)$ using his own decryption key SK_B:

$$m = \frac{Z^r \cdot m}{(Z^{br})^{\frac{1}{b}}}$$

If the original user Alice wished to decrypt the message, then a similar process would unfold; the manager would create a re-encryption key $RK_{P\to A}$ and Alice would decrypt her ciphertext $E_a(m)$ using her decryption key SK_A. Thus, the manager can allow any user within the group to access the encrypted data stored within the cloud. Here, *first-level encryption* is demonstrated [8], where the content $E_b(m)$ available from the manager may be decrypted only by the holder of SK_B; the content may not be re-encrypted a second time and read by a third party such as user C in U_P. If C requires access, then the use of $RK_{P\to C}$ to carry out a re-encryption of $E_p(m)$ to $E_c(m)$ is required.

Data Flow

To summarize, the flow of ciphertext in the system between two users is as follows:

$$\boxed{A} \xrightarrow{E_p(m)} \boxed{P} \xrightarrow{E_p(m)} \boxed{M} \xrightarrow{E_b(m)} \boxed{B}$$

The cryptographic operations explained in this section are shown visually in Table 4.2.

Step	**Alice** (A)	**Cloud** (P)	**Manager** (M)	**Bob** (B)
1			Computes $PK_p = g^p$ and $SK_p = p$, and shares PK_p with cloud. Similarly, computes $SK_B = b$ and sends it to B.	
2	Obtains PK_p from cloud, picks random r, encrypts m as $E_p(m) = (Z^r \cdot m, g^{pr})$, and sends it to the cloud.			
3		Stores $E_p(m)$, and sends a copy of it to M on request.		
4			Computes $RK_{P\to B} = g^{\frac{b}{p}}$. Re-encrypts $E_p(m)$ as $E_b(m) = (Z^r \cdot m, Z^{br})$.	
5				Downloads $E_b(m)$ from M and decodes $m = \frac{Z^r \cdot m}{(Z^{br})^{\frac{1}{b}}}$ using SK_B.

Table 4.2: A summary of operations in manager-based re-encryption.

Key Re-generation

If a new user Charlie, or C, joins the group, then he registers with the manager which grants authorization, and is given a decryption key SK_C. C will be able to receive and decrypt only the content that the manager is willing to re-encrypt for him, as ciphertext $E_c(m)$. If Charlie leaves the group, then the manager removes him from its access list; it will no longer re-encrypt data for C on retrieval attempts.

4.3.3 Discussion

An important advantage of this model lies in is its elimination of expensive key re-generation and re-distribution for all users whenever group membership changes. It preserves data confidentiality for the client; data in the cloud remains encrypted and unreadable in its original form by the provider at all times. For a new user that joins the group, the manager can choose to decrypt data stored only after a certain time, hence providing *backward secrecy*. For a user that leaves the group, and whose access is revoked, none of the stored data can be decoded independently by that user, hence providing *forward secrecy*.

Unlike the encrypted file storage mechanism described in [8], the proposed proxy re-encryption is applied to the ciphertext itself rather than a container (i.e. lockbox) for decryption (i.e. content) keys, so that decryption keys are not shared by multiple users which carries a risk of compromise or collusion; additionally, the container itself is not required to be downloaded by users as extra overhead. Additionally, unlike in the related work, the client does not need to make a separate request against an access control server for every single data fetch in order to request a re-encryption, and then make a second request to actually download the data of interest. Conservation of communication in this manner is important for mobile device users.

Some opportunities arise for increasing performance. The manager may cache the most recently re-encrypted content for each user so that multiple accesses of the same data by the same user may be serviced more quickly. A replacement strategy such as *least-recently-used* may be employed; if the same user requests the same records repeatedly, then re-encryption would not need to be re-done on a cache hit. In all cases, the recipient completes only a single decryption operation, which is suitable for a resource-constrained user. Additionally, the manager can take on additional responsibilities if allowed by the

system model. If there is a secure link between the users and their manager (through a VPN connection for instance, or if all user entities are connected on an intra-net behind a secure firewall), then users may communicate freely with the manager without the need for additional data encryption in transit in the final leg. In this case, the authority can manage all of the encryption and decryption needs of its members, thereby unloading that processing burden from lightweight mobile device users.

A very significant disadvantage of this approach, however, is that for each retrieval attempt of a new data block or record, the manager must perform re-encryption using an asymmetric key. A bilinear pairing operation based on a Weil and Tate pairing is several times more costly than a scalar multiplication [64]. Although it is an expensive operation, it can be accomplished in the private portion of a hybrid cloud if the manager is a component of it, thus taking advantage of its scalability. A mechanism for using the public portion of a cloud, which would typically scale much more easily for this purpose, will be described in the following section. Another disadvantage is that, because the manager stores all decryption keys, it must be fully trusted; hence, it is a point of vulnerability. Furthermore, it requires full trust by the client which may be unrealistic in some systems.

The proposed scheme is presented in this chapter as a plausible and straightforward adaptation of the concept of re-encryption cryptography to a cloud computing context. However, its stated disadvantages render it a non-optimal key management scheme for mobile cloud computing. A more practical scheme is presented in the following Section 4.4.

4.3.4 Novel Variants

Notwithstanding the fundamental scalability problem of the manager, additional optional variants are presented to improve the security or performance of the original scheme:

Encryption Using an Owner Key

In order to reduce the cost of re-encryption for all requests, the protocol may be modified so that rather than using the partition key PK_P for encryption, user A would use her own public key PK_A, and upload ciphertext $E_a(m)$ to the cloud. Upon data retrieval, the manager would be required to perform re-encryption for another user, such as B, using re-encryption key $RK_{A \to B}$ supplied by M:

$$RK_{A \to B} = g^{\frac{SK_B}{SK_A}} = g^{\frac{b}{a}}$$

This technique would allow Alice to retrieve data directly from the cloud that she could then decrypt without the aid of the manager, which has good practical application; in many conceivable use cases, it would be expected that the same user that uploaded data would be the one that would most frequently access it. The trade-off is that in case A was to leave the group, the manager would need to invalidate all data uploaded by A; one option would be for the cloud provider to re-encrypt it to the partition key, and then control all access to it from that point going forward; even so, no key re-distribution would occur.

Optimized Manager-Based Re-Encryption

If the manager introduces too much latency into the system due to its workload, it is possible to substantially reduce its communication and processing burden by transferring some of it to the users; this is particularly effective if system usage typically entails repeated fetches of the same data, as is generally the case. The manager's critical role in the protocol described thus far is to perform the re-encryption task for every fetch of ciphertext by the same user. However, observe that the $Z^r \cdot m$ subcomponent of the encrypted ciphertext $E_p(m)$ stored in the cloud is not directly involved in this operation; it may be directly downloaded from the cloud by the recipient B, who will then await the second component Z^{br} from M. In this way, M avoids the overhead of fetching $E_p(m)$ in its entirety from the cloud. Furthermore, M will reduce the number of pairing operations that it needs to perform in the original re-encryption scheme in order to compute the Z^{br} component, and will thus significantly improve its scalability. The modified scheme proceeds as follows:

The data owner first uploads $E_p(m)$ to the cloud, which stores it, as before. When the recipient B first makes a request of the ciphertext $E_p(m) = (Z^r \cdot m, g^{pr})$, it contacts the cloud and manager jointly to receive the ciphertext components that it needs for the decryption. The provider first provides the component $Z^r \cdot m$ to B to fulfill its obligation in the client request. As part of the transaction, the provider also sends the component g^{pr} to the manager, which performs a single pairing operation on it, utilizing SK_P, as follows:

$$Z^r = e(g^{pr}, g^{\frac{1}{SK_P}}) = e(g^{pr}, g^{\frac{1}{p}})$$

The manager stores the result Z^r for future use in re-encryptions for any user. Note that this operation by the manager occurs the first time that the ciphertext in question is accessed by any user, and the same component can be re-used for any user in the future. Next, to process the current outstanding request from B in particular, it exponentiates the result Z^r using a copy of B's decryption key SK_B, to obtain the result Z^{br}:

$$Z^{br} = (Z^r)^{SK_B} = (Z^r)^b$$

The manager then sends the result Z^{br} as the second required component of the ciphertext to B, so that B now possesses $E_b(m)$ in its entirety and can perform the decryption, as described previously. The cryptographic operations explained in this variant are shown visually in Table 4.3.

These modifications to the manager-based re-encryption scheme significantly improve the performance of the manager, which now only carries out a single pairing operation for each ciphertext and user. Repeated subsequent requests of the same data result in a single exponentiation by the manager, instead of an expensive pairing operation as in the original scheme. The only added cost is an additional component Z^r to be retained for each ciphertext, but it is applicable to all users. Furthermore, to reduce the communication cost for the mobile user to a single request, it is possible for the manager to fetch the $Z^r \cdot m$ component from the cloud and provide it directly to the user; the cost is added communication for the manager itself, but it is no worse than in the original protocol.

An undesirable side effect is that if B leaves the group, he can continue to download and access encrypted data in the cloud. This issue is solved in the model presented in the next section, in which the cloud data undergoes a transformation that prevents this possibility. Also, B must initiate download requests to both the cloud provider and the manager; in the next model, downloads only involve the cloud provider.

Table 4.3: A summary of operations in a variant of manager-based re-encryption.

Step	**Alice** (A)	**Cloud** (P)	**Manager** (M)	**Bob** (B)
1			Computes $PK_p = g^p$ and $SK_p = p$, and shares PK_p with cloud. Similarly, computes $SK_B = b$ and sends it to B.	
2	Obtains PK_p from cloud, picks random r, encrypts m as $E_p(m) = (Z^r \cdot m, g^{pr})$, and sends it to the cloud.			
3		Stores $E_p(m)$, and sends the g^{pr} component of it to M on request.		
4			On the first (and only) request for $E_p(m)$ from any user, and from the component g^{pr}, computes $e(g^{pr}, g^{\frac{1}{p}}) = Z^r$, and stores it.	Requests the ciphertext components from P and M.

Table 4.3: (continued)

Step	Alice (A)	Cloud (P)	Manager (M)	Bob (B)
5		Sends the $Z^r \cdot m$ component of $E_p(m)$ to B on request.	On the first and all subsequent requests by B in particular, and using SK_B, computes $Z^{br} = (Z^r)^{SK_B} = (Z^r)^b$ and sends it to B.	
6				Downloads all components of $E_b(m) = (Z^r \cdot m, Z^{br})$ jointly from the cloud and M, and decodes $m = \frac{Z^r \cdot m}{(Z^{br})^{\frac{1}{b}}}$ using SK_B.

Limited Trust of the Manager and Data Owner

In the scheme presented, the manager retains the decryption keys of all users, as well as the partition secret key, and is assumed to be wholly trusted by the entire user population. This assignment of trust has utility in regenerating and redistributing a decryption key in the case where a mobile user loses it due to a device loss or erasure, or in automatically regenerating a key due to time expiry in compliance with an IT security policy.

However, to restrict access to data of the most sensitive nature, and mitigate the consequences of an attack on the manager, it may be desirable for a user to generate his or her own *decryption* key, instead, and not share it with the manager; fortunately, the re-encryption task performed by the manager is not hampered as a result of this restriction.

During the key generation phase, Alice (A) may randomly select a secret key $SK_A = a \in \mathbb{Z}_q^*$ and compute a public key $PK_A = g^{SK_A} = g^a$, to be used for encryption; the public key is uploaded to the manager; it does not necessarily require an out-of-band transport for security. The secret decryption key $SK_B = b \in \mathbb{Z}_q^*$ for recipient Bob (B) is independently generated by Bob himself, and the public key $PK_B = g^b$ is shared with the data owner A for re-encryption purposes. Similarly to the cloud provider, at no time can the manager decrypt any user data stored in the cloud, as it has no access to either private key. Furthermore, Bob does not share his private key with Alice.

Later, during a re-encryption operation, the data owner exponentiates the public key PK_B with the inverse of SK_A to calculate the re-encryption key $RK_{A \to B}$, achieving the same result as before, where the ciphertext is transformed to a version that can be decoded by B:

$$RK_{A \to B} = (PK_B)^{\frac{1}{SK_A}} = (g^{SK_B})^{\frac{1}{SK_A}} = g^{\frac{SK_B}{SK_A}} = g^{\frac{b}{a}}$$

The manager still computes the expensive pairing operation entailed in the re-encryption task. In this way, highly sensitive data cannot be read by a compromised manager, unlike in [8], where the access control server must be fully trusted. The cost is additional key generation and distribution demands placed on the data owner.

4.4 Cloud-Based Re-Encryption

4.4.1 Introduction

A potential problem with the manager-based encryption scheme is that the manager is allocated all re-encryption tasks, and its ability to scale may be limited. An alternative and novel model is now presented, where the cloud provider is delegated the responsibility of re-encryption, in order to leverage its advantages in computational capacity. The manager still exists in this scenario, playing the role of key coordinator; however, it is no longer a bottleneck for re-encryption operations in the system. All data re-encryption operations are handled by the cloud provider, which is highly scalable.

4.4.2 System Operation

Key Generation and Encryption

Refer to Figure 4.2. As before, in the setup phase, the manager M generates version 0 of a public and private key pair, PK_{P_0} and SK_{P_0} for the data partition P, in a similar manner to what was described in Section 4.3; it then distributes a copy of PK_{P_0} to all current authorized users in the user group U_P, including A and B. Alternatively, PK_{P_0} may be stored in the public key directory accessible to all users. The secret partition key is never shared with the cloud provider. M directly distributes SK_{P_0} to all of its current users who are entrusted with the safekeeping of it.

Once again, user A, or Alice, wishes to store encrypted data in the cloud. A encrypts a message m with PK_{P_0}. A then uploads the ciphertext $E_{p_0}(m)$, and any optional associated policy settings, to the cloud provider:

$$E_{p_0}(m) = (Z^r \cdot m, g^{p_0 r})$$

The data is stored in P, in encrypted form.

Decryption

User B, or Bob, is another user in the same group as A, and requests the data $E_{p_0}(m)$ that A has uploaded to partition P. Since B has a copy of the secret partition key SK_{P_0}, he can decrypt the data:

$$m = \frac{Z^r \cdot m}{(Z^{p_0 r})^{\frac{1}{p_0}}}$$

Both A and B receive SK_{P_0} during the set-up phase from the cloud provider. A may also provide it directly to B in peer-to-peer fashion, over a secured Bluetooth channel, for instance; a local link such as this would not incur the same high transmission cost as a 3G or 4G wireless channel. All users in U_P may retrieve the message uploaded by A to the cloud, by directly obtaining $E_{p_0}(m)$ from the cloud provider, and using the same shared decryption key SK_{P_0}. Thus, in this model, *second-level encryption* is demonstrated [8]; as applied here, the ciphertext published by the cloud may be decrypted by a recipient who holds the original secret partition key; additionally, a re-encryption round on the ciphertext is possible by the provider acting as a *delegate*, which will transform it into a first-level ciphertext so that it may be decrypted only by the holder of a newer partition key.

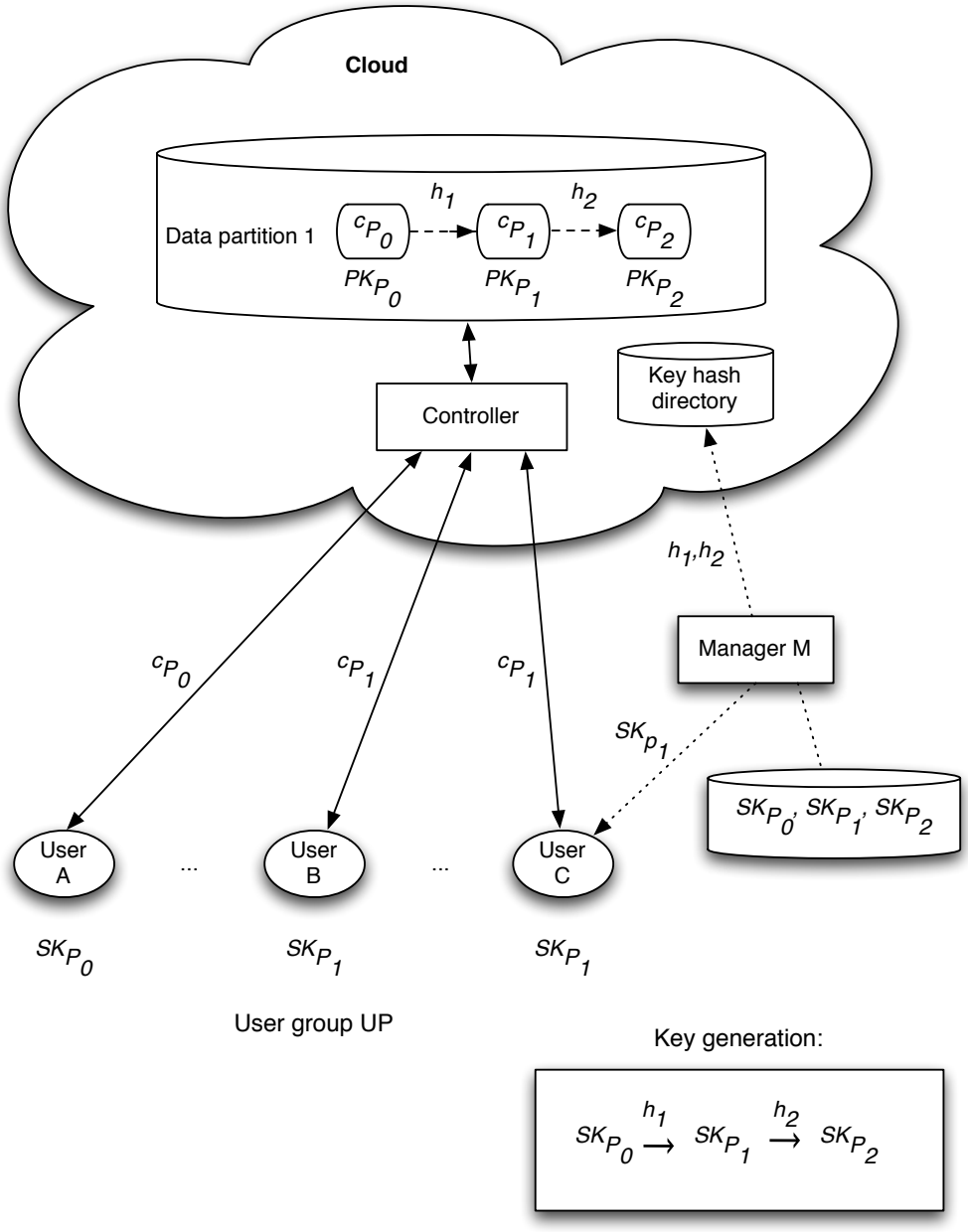


Figure 4.2: A model of key management using cloud-based re-encryption.

Re-Encryption

If a new user Charlie, or C, joins the group and the manager authorizes him, then the present partition key PK_{P_0} is invalidated; it becomes obsolete, and a new version of the key must be generated. M first authorizes C, approving membership. The manager then creates a new random salt, with value h_1, and adds it to the key SK_{P_0}; it then hashes the result through a secure hash such as SHA-2, to generate the new (version 1) key SK_{P_1}. In general:

$$SK_{P_v} = p_v = f(SK_{P_{v-1}}, h_v)$$

for version $v = 1, \ldots, n$, random $h_v \in \mathbb{Z}$ and secure hash function f. The public key PK_{P_v} is then derived from the secret key SK_{P_v}, as before: $PK_{P_v} = g^{p_v}$.

The hash value used to generate the new key is then shared with all current authorized users in the group. The entire hash chain $H = \{h_x | x \in \mathbb{N}, x \leq y\}$, where y is the current version number corresponding to the most recently created key, can be stored in the cloud and shared with authorized users in U_P; the random hash input values themselves are insufficient for the cloud provider to determine the key. The newly joined user C will be unable to decrypt the message already stored by A; it was encrypted with an older key, with a value less than y.

The accessibility of the ciphertext by C may be dependent on the default policy, or an optional custom policy originally attached to the data by A. By default, it may require that the data $E_{p_0}(m)$ presently stored in the cloud partition be re-encrypted with the new partition key. If the policy rule requires permission from A to accomplish this, then C will be unable to decode the data until it is given. The re-encryption need not necessarily occur at the time of C's admission into the group; it may be triggered at the time of his data access attempt. It may also be requested by the manager or any other authorized user at any time, i.e. when that data is next accessed. If the data is re-encrypted by the cloud provider using h_1 to form ciphertext $E_{p_1}(m)$, then it can be decoded by C using the new key SK_{P_1}, where $y = 1$.

To re-encrypt the message, the cloud provider requires knowledge of the re-encryption key that is based on the latest version of the private partition key; this re-encryption key is generated and provided by the manager as soon as the key is updated. The re-encryption key $RK_{P_0 \to P_1}$ is a transformation from SK_{P_0} to SK_{P_1}:

$$SK_{P_1} = p_1 = f(SK_{P_0}, h_1) = f(p_0, h_1)$$

$$RK_{P_0 \to P_1} = g^{\frac{SK_{P_1}}{SK_{P_0}}} = g^{\frac{p_1}{p_0}}$$

During re-encryption, ciphertext $E_{p_0}(m)$ is transformed into $E_{p_1}(m)$:

$$\begin{aligned} \text{From } E_{p_0}(m) &= (Z^r \cdot m, g^{p_0 r}), \\ \text{Compute: } e(g^{p_0 r}, RK_{P_0 \to P_1}) &= e(g^{p_0 r}, g^{\frac{p_1}{p_0}}) = Z^{p_1 r} \\ \text{Publish: } E_{p_1}(m) &= (Z^r \cdot m, Z^{p_1 r}) \end{aligned}$$

C can now proceed to download and decrypt the message:

$$m = \frac{Z^r \cdot m}{(Z^{p_1 r})^{\frac{1}{p_1}}}$$

The cloud provider stores a history of the key versions, including the version number of each key, the public partition key itself, the corresponding re-encryption key required to re-encrypt the original uploaded ciphertext to the corresponding new version, and the hash value used to create the re-encryption key, as illustrated in the following versioning array:

$$\begin{bmatrix} 0 & PK_{P_0} & - & - \\ 1 & PK_{P_1} & RK_{P_0 \to P_1} = g^{\frac{p_1}{p_0}} & h_1 \\ 2 & PK_{P_2} & RK_{P_0 \to P_2} = g^{\frac{p_2}{p_0}} & h_2 \\ \vdots & \vdots & \vdots & \vdots \\ y & PK_{P_y} & RK_{P_0 \to P_y} = g^{\frac{p_y}{p_0}} & h_y \end{bmatrix}$$

Note once again that the cloud provider can never decrypt and view the original contents of the message, as the original key SK_{P_0} in the chain is unknown. Each new re-encryption corresponds to a new and higher version number. Each new key is traceable to a version number, so that any user may determine whether the key required to decrypt the ciphertext is in his or her possession. If not, when the client requests the ciphertext from the cloud provider, he or she can request that it be re-encrypted to the same version of the key that is actually in the user's possession, if the ciphertext is encoded with an earlier version and backward secrecy is not enforced. On the other hand, if the ciphertext version is more recent, then the user can re-assemble the correct private key using the hash value chain history H that can be downloaded at any time from the cloud; the user must then perform only a single decryption; multiple decryptions are not required.

At the latest, the stored data needs to be re-encrypted when access to it is attempted; the effect of this is that re-encryption will only occur on the most frequently-accessed data. Whenever a fetch request for cloud data is made, the cloud provider first checks whether the message version matches the version of its most recent key in possession, and performs re-encryption if it does not.

If C leaves the group, then the manager will increment the key version, re-generate the partition key, and inform the server that re-encryption is required. C will not be issued any further key updates; he will no longer be authorized to access the key hashes stored within the key hash directory on the cloud, or request them from the manager. To guarantee that the cloud cannot collude with C and reveal the hash history even when C fails authentication, the hash history may either be published by the manager only, or the manager may periodically reset the hash history and designate a new starting key version 0 while still allowing the history to reside in the cloud.

Data Flow

To summarize, the flow of ciphertext in the system between two users is as follows, with the manager no longer playing the role of an intermediary in the communication:

$$\boxed{A} \xrightarrow{E_p(m)} \boxed{P} \xrightarrow{E_p(m)} \boxed{B}$$

The cryptographic operations described in this section are summarized in Table 4.4.

Table 4.4: A summary of operations in cloud-based re-encryption.

Step	**Alice** (A)	**Cloud** (P)	**Manager** (M)	**Bob** (B)	**Charlie** (C)
1	Obtains $PK_{P_0} = g^{p_0}$ from the cloud provider, picks random r, encrypts m as $E_{p_0}(m) = (Z^r \cdot m, g^{p_0 r})$, and sends it to the cloud.				
2		Stores $E_{p_0}(m)$ and its associated version 0.			
3				Downloads $E_{p_0}(m)$. Receives $SK_{P_0} = p_0$ from M and uses it to decode $m = \frac{Z^r \cdot m}{(Z^{p_0 r})^{\frac{1}{p_0}}}$.	
4			Authorizes new member C. Computes $RK_{P_0 \to P_1} = g^{\frac{p_1}{p_0}}$ and sends it to P. Sends $SK_{P_1} = f(p_0, h_1) = p_1$ to C.		Receives SK_{P_1} from M.

Table 4.4: (continued)

Step	Alice (A)	Cloud (P)	Manager (M)	Bob (B)	Charlie (C)
5		Re-encrypts $E_{p_0}(m)$ as $E_{p_1}(m) = (Z^r \cdot m, Z^{p_1 r})$ using $RK_{P_0 \to P_1}$, and updates version to 1.			
6					Downloads $E_{p_1}(m)$ and decodes $m = \frac{Z^r \cdot m}{(Z^{p_1 r})^{\frac{1}{p_1}}}$ using SK_{P_1}.

4.4.3 Discussion

The cloud-based re-encryption model off-loads the processor-intensive task of re-encryption to the cloud provider. It is consistent with the underlying assumption behind a cloud computing system: that it can scale to a much greater degree than its client can in terms of computational ability. Crucially, unlike in the scheme described in the previous section, the manager is *not* involved in each data fetch operation; it is only occasionally involved in creating new keys when new users join. Another advantage is that the re-encryption task may be executed only when necessary; it is only required at most once for each data record whenever group membership changes. The re-encryption tasks may be batched and executed during off-peak hours, or may be done only when a new fetch of the record is made, at the latest. This model permits more direct access to the cloud while allowing all security requirements to continue to be satisfied. Any authorized user can write and read encrypted data directly to and from the cloud without involvement of the manager or any other proxy, resulting in fast access on a regular basis. Data confidentiality is preserved in

this model even when changes to group membership occur. Since a new user is only given the latest iteration of a key and cannot decrypt messages encrypted earlier with older keys, backward secrecy is preserved (however, if this security feature is deemed unimportant, then re-encryption is not necessary in the case where user membership increases). The reciprocal is that a user that leaves the group is no longer issued key updates. Since re-encryption occurs prior to a new data fetch request, the user is no longer able to decrypt data; forward secrecy is preserved. User memberships tend to increase in practice, however.

The use of hashes as public key material makes it unnecessary to distribute a new version of the partition key to all users when it becomes re-generated by the manager. The history of re-encryption keys can be stored with the encrypted data and made available to all users by the cloud provider; it can be downloaded along with the ciphertext. An existing user will be able to generate the partition key by knowing the hash value history; the cost of re-distribution of keys on every change in membership is avoided. Storage requirements for each user are modest; it is unnecessary to store the original key and the entire history of hash values. On a key re-generation, each user can use his or her hash values to arrive at the latest key, and discard all of its history. Thus, only one secret key must be locally stored for each partition that the user interacts with.

Note that the original re-encryption protocol based on BBS [18] allowed the same encrypted content to be re-encrypted multiple times by the cloud provider; the cost of this in the proposed protocol is that it would allow transitivity of delegations. For example, it would allow the cloud provider to derive its own re-encryption key $RK'_{P_x \to P_{x+2}}$ based on public key PK_x to PK_{x+2} as follows:

$$RK'_{PK_x \to PK_{x+2}} = RK'_{PK_x \to PK_{x+1}} \times RK'_{PK_{x+1} \to PK_{x+2}} = \frac{p_{x+2}}{p_x}$$

This flexibility would allow the cloud provider to retain only the most recent re-encryption from the newest available key, and to keep re-encrypting it multiple times as the key evolved through a process of *delegation*. In this case, $E_{p_{x+1}}(m)$ would be re-encrypted directly to $E_{p_{x+2}}(m)$, rather than from the original $E_{p_x}(m)$. The cost is that it would allow a newly joined user to collude with the holder of SK_{x+1} and the provider by sharing its private key SK_{x+2}; the cloud provider could deduce $RK'_{PK_{x+1} \to PK_{x+2}}$, as shown, and re-encrypt data for the new user that was not actually intended to be accessed by him. In contrast, the re-encryption protocol based on bilinear maps, as described here, is not transitive, and thus such delegation to new users is not allowed without arbitration from the manager. The

protocol is collusion-safe, as discussed in [8]; a user that knows $SK_{p_1} = p_1$ cannot collude with the cloud provider, which knows $RK_{PK_{p_0} \to PK_{p_1}} = g^{\frac{p_1}{p_0}}$, and recover $SK_{p_0} = p_0$. This protection is at the expense of having to retain the original ciphertext $E_{p_0}(m)$ in the cloud for use in all future re-encryptions, and to incur a storage cost. The provider may still cache the ciphertext resulting from the most recent re-encryption for immediate access.

The main drawback with this approach is the re-encryption task required whenever group membership changes, which is a relatively expensive operation. Unlike the previous model, it is performed within the cloud, however, which has the ability to instantly scale to meet the processing demand. Also, there still exists the risk of the key being illegitimately shared by a misbehaving (yet authorized) user with that of an unauthorized one. All users are inherently entrusted with the secret partition key, unlike in the previous manager-based re-encryption scheme. The cloud provider can perform user authentication against its ACL as a fallback mechanism, however.

4.4.4 Variant

User-Generated Keys

It is possible to restrict the scope of trust of the manager for highly-sensitive user data. In a variant of this model, as opposed to employing a manager-generated initial partition key P_0, A herself may generate the key pair PK_{P_0} and SK_{P_0}. These keys may then be used for the first encryption of a data record that is uploaded to the cloud. The advantage of this approach is that A can then completely control access to that data record by creating new re-encryption keys based on the manager-created hashes. The manager will never be able to read the data, and thus does not have to be trusted to the same degree as in the standard case described above. The manager will only generate and issue new re-encryption keys to all authorized users for subsequent versions; the manager will never obtain a copy of the first-version key so that it can reconstruct the key history and be in a position to decrypt all data in the partition uploaded by A. For instance, A can simply share the component $g^{\frac{1}{P_0}}$ with M, and M will then exponentiate it with P_1, which it generated, to obtain the re-encryption key $RK_{P_0 \to P_1} = g^{\frac{P_1}{P_0}}$.

The granularity of access control may be controlled by the user; A may generate a secret key pair for each new data record created, or the same key pair for all records. The

cost of this approach is that A must share her keys with all users who require read access to the data. In a mobile scenario, this may be accomplished by A pairing with another user via Bluetooth in peer-to-peer fashion to avoid the cost of wireless 3G or 4G transfer, as only a small one-time transfer of key material is needed.

4.5 Evaluation and Implementation of Models

4.5.1 Qualitative Cost Comparison

The processing, storage, and communication costs of the transactions in the two proposed re-encryption models are shown in Table 4.5. The main advantage of these models is that constant key re-generation need not occur between the cloud (or a proxy) and the user set. Considering that the user base will largely comprise mobile device users, the conservation of wireless communication exchanges is significant and valuable. The trade-off is in the automatic and continuous re-encryption necessary as user memberships naturally evolve. In the cloud-based re-encryption model, the partition key is generated by the manager, but the re-encryption itself is carried out by the cloud provider; importantly, fetching data from the cloud does not involve the manager as an intermediary in each data fetch session. The proxy re-encryption model requires the manager to perform re-encryption on each client request, however, and so it is not as scalable in a cloud context; it still has reasonable potential if the manager is situated inside of a private cloud.

4.5.2 Performance Measurement

In order to understand the execution cost of the protocol on real hardware, the cloud-based re-encryption algorithm described in the previous section was implemented in Java using jPBC (Java Pairing-Based Cryptography Library) version 1.2.1 [33], a porting of the PBC (Pairing-Based Cryptography Library) in C [76]. The encryption, re-encryption, and decryption tasks, as described earlier, were timed on different platforms; portability was provided by Java 6 Standard Edition. The desktop platform consisted of an Apple iMac with a quad-core 64-bit 3.4 GHz Intel Core i7 processor and 16 GB of RAM, running Mac OS X 10.8.2 (Mountain Lion). The smartphone platform consisted of a Google Nexus

Computational complexity		
Description	*Proxy-based re-encryption*	*Cloud-based re-encryption*
Key generation (manager)	$BP + E$	$BP + E$
Key generation (user)	-	-
Encryption (user)	$2 \cdot E + M$	$2 \cdot E + M$
Decryption (user)	$E + M$	$E + M$
Key re-generation (manager)	-	$H + E$
Key re-generation (user)	-	-
Re-encryption (server)	-	BP
Re-encryption (manager)	$BP + E$	E
Computational costs		
Description	*Proxy-based re-encryption*	*Cloud-based re-encryption*
Key generation	(none)	(none)
Re-encryption	1 per join/leave	1 per join/leave
	(operation done by proxy)	(operation done by cloud)
Access model		
Description	*Proxy-based re-encryption*	*Cloud-based re-encryption*
Data fetch	Via proxy	Direct-from-cloud
Storage costs		
Description	*Proxy-based re-encryption*	*Cloud-based re-encryption*
Key storage	All stored in manager	All stored in manager

Table 4.5: The processing, storage, and communication costs of the re-encryption models. The cryptographic operations include: Hashing (H), Exponentiation (E), Bilinear pairing (BP), and Multiplication (M).

One phone with a single-core 1 GHz Qualcomm QSD 8250 Snapdragon ARM processor and 512 MB of memory, running Android OS 2.3.6 Gingerbread.

The cloud platform consisted of a single Google App Engine (GAE) web application instance. The reference for billing is a front-end instance comprising a 1.2 GHz Intel x86 processor with 128 MB RAM, billed at 10¢ per hour; the actual number of CPU cycles used is internal to the App Engine and not exposed. The cloud servlet application posted responses to HTTP requests.

For the purpose of experimentation, Google allows a free test account for use with up to 500 MB of storage and up to 5 million page views per month. The Google App Engine SDK supports the URL Fetch service only; it does not support sockets for communication, for security reasons. It also constrains some aspects of OS functionality by disallowing processes, threads, dynamic library loading, and writing to the data store. Nevertheless, it is a highly popular cloud development platform.

A *Type A pairing* was utilized in the algorithm, which is the default curve in the PBC library's included test code; the group order was 160 bits long, and the base field order was 512 bits long, which is suitable for cryptographic use. The aggregate timing results obtained from experiments on each of the three platforms are shown in Table 4.6.

Overall, the re-encryption task was found to be much more feasible on a cloud instance or a fast desktop computer; in the latter case, it was over 50 times faster than on the smartphone. Although the re-encryption task may be performed on a scalable server, the advantage of off-loading it to the cloud is that it can scale almost without bound. Additionally, GAE provides faster back-end instances with up to a 4.8 GHz CPU and 1 GB memory. The performance attained on the smartphone was reasonable; the encryption task by the data owner was accomplished in approximately 1.5 seconds, which is a one-time cost, and the decryption task by the recipient in only approximately 0.1 seconds. The presented benchmark results were achieved using libraries that are not yet highly mature and optimized for a mobile platform. However, they serve to validate the comparative strengths of mobile devices and clouds, as described, which the re-encryption model leverages.

Platform	**Task**	**Timings (in ms)**				
		μ	σ	s	LB	UB
Smartphone (Nexus One)	Encryption by A	1,505.2	40.5	1,636.9	1,425.9	1584.5
	Re-encryption	979.8	37.3	1,393.2	906.6	1,052.9
	Decryption by B	107.4	28.1	790.1	52.3	162.5
Desktop (iMac)	Encryption by A	32.9	2.1	4.3	28.9	37.0
	Re-encryption	19.1	1.1	1.1	17.1	21.2
	Decryption by B	1.1	0.2	0.0	0.7	1.5
Cloud (Google App Engine)	Encryption by A	426.6	49.1	2414.6	330.3	522.9
	Re-encryption	260.8	42.8	1828.8	177.0	344.6
	Decryption by B	13.7	23.9	570.3	0	60.5

Table 4.6: The performance results obtained from the re-encryption implementation. The following symbols are used: μ is the sample mean, σ is the sample standard deviation, s is the sample variance, and LB and UB are the lower and upper bounds, respectively, of the 95% confidence interval, for each set of runs on each platform. 100 runs were executed on each of the smartphone and desktop platforms, and 50 runs were executed on the cloud platform (so as not to trigger a request timeout). The three consecutive operations shown for each platform are: encryption time by A using the P_0 key, re-encryption time from P_0 to P_1 keys with pairing, and decryption by B using the P_1 key. All operations were performed on a 48-bit data block.

4.6 Summary

Cryptographic protocols based on data re-encryption have been adapted to a cloud computing system model in order to gauge their viability in improving communication security and supporting highly scaleable and secure cloud computing applications serving an extremely large mobile device user population. Appropriate modifications have been proposed to support both public and private clouds, and standard 3G or 4G wireless as well as peer-to-peer links, in order to reduce the cost of communication for mobile users securely accessing and storing data in a cloud for dissemination purposes to a large reader population.

The manager-based re-encryption scheme addresses the cost of re-keying operations in a cloud-based key management protocol by having a trusted authority, independent of the cloud provider, perform re-encryption before delivering a request to the client. The authority becomes the gateway for data access to the cloud; in doing so, it does not necessitate any key updates over time. It is particularly suitable for a private cloud environment, but entails a considerable computational load. A novel protocol based on data re-encryption has been proposed to offer higher scalability and to support an extremely large mobile device user population. This is achieved by leveraging the cloud provider's scalability to perform the required re-encryption tasks inside the cloud itself, rather than inside the manager; at the same time, this occurs without granting the cloud provider access to sufficient key material to decode the user data. The manager, as a trusted authority is only responsible for key re-generation; the evolving key material to construct iterations of secret keys can be securely shared through the cloud provider itself, resulting in a more efficient and scalable security protocol. The scheme ensures that the cloud provider can never read user data, but can nevertheless transform it through re-encryption to efficiently manage access for users that continuously join and leave the system.

Chapter 5

Hybrid Attribute- and Re-Encryption-Based Key Management

5.1 Introduction

A HYBRID SCHEME entailing attribute-based encryption and optional group keying techniques is proposed in this chapter, such that computationally-intensive work is performed by the cloud provider or a trusted manager rather than the mobile data owner. The re-encryption system model described in Section 3.1.5 on page 37 and the adversary and threat model described in Section 2.4.1 on page 19 are assumed.

Related work on attribute-based encryption is described in Section 5.2, beyond what is presented in Chapter 3 on page 30. In Section 5.3, the proposed algorithm for attribute-based encryption and re-encryption suitable for mobile users of the cloud is given. In Section 5.4, optional features of the algorithm such as delegation are presented, for completeness. In Section 5.5, the algorithm is assessed for its usefulness in a mobile cloud computing system. In Section 5.6, the results of an implementation of the proposed scheme on actual mobile devices and an operational cloud system are presented and discussed; a simulation is then used to demonstrate its scalability potential. Finally, Section 5.7 provides concluding remarks.

The content of this chapter is based on work that has been published [111].

5.2 Related Work on Attribute-Based Encryption

The technique of Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [17] offers numerous advantages in the envisioned target environment. It allows a user to obtain access to encrypted data in the cloud based on the possession of certain attributes that satisfy an access structure defined in the cloud, rather than the possession of a particular individual or group key that must be disseminated to all interested parties in advance. The requisite attributes may be determined by a data owner in advance; this owner is responsible for generating the user data to be shared, encrypting it, and uploading it to the cloud. Unauthorized access to stored data is not in itself an issue due to the protection afforded by CP-ABE. Furthermore, the data owner is not required in every data transaction involving other users, which is advantageous in the case where *always-on* connectivity cannot be guaranteed. It is impossible for any two users to collude by combining their individual attributes to gain access that would otherwise not have been individually granted. Normally, a scheme based on CP-ABE relies upon the data owner granting access permission through an access tree, which requires his or her constant availability. Some works have modified CP-ABE so that key material is distributed among multiple parties; for instance, a data owner and a trusted authorizer may function in concert to grant access permission to other users, building on the OAuth standard [106]; the solution, however, is not tailored for a mobile environment due to its computational demands, the required constant availability of the data owner, and time-based expiration of access leading to frequent key retrieval.

Revocation of an authorized user is particularly difficult to accomplish efficiently in CP-ABE and is usually addressed by extending attributes with expiration dates or by an authority distributing keys with expiration dates [17]. In some cases, a tree of revocable attributes may need to be maintained and a trusted party assigned to validate the revocation statuses of users; the access control may be system-wide or more fine-grained. A revocation mechanism using linear secret sharing and binary tree techniques, where each user is associated with an identifier on a revocation tree, is one example [73]. The difficulty with this general approach in a mobile context is that it results in mobile users having to incur the communication cost of continually requesting new keys, while wireless communication always remains expensive. Also, the data owner is typically a mobile user as well,

and thus the owner cannot effectively manage access control on demand for other users due to its transient connectivity. Revocation for data outsourcing purposes has been proposed that relies on stateless key distribution and access control on the attribute level, but requires a trusted authority and encumbers the data owner with a pairing operation [54], a cryptographic function that is very computationally expensive.

As a next step in the evolution of such techniques, proxy re-encryption has been combined with CP-ABE [74] such that re-encryption keys are computed by the cloud provider based on a secret that is pre-shared between the data owner and the provider, as well as the provider's internal clock. The re-encryption keys must be computed for all attributes in the access structure, which could be very numerous. Another idea is to securely embed the data key within the header of the record stored in the cloud [37]; a privileged manager group is responsible for generation of re-encryption keys, but it must also distribute the secret header key to the recipient to complete the process. A different approach has been suggested where attribute revocation events occur, and in response, an authority redefines master key components for the attributes, user secret keys are updated to a new version, and data is re-encrypted by the proxy server [120]; the difficulty is that revocation is dependent upon modifications to attributes, resulting in costly key updates on each revocation, and the proxy server must be given access to user attribute information. Finally, a technique that combines CP-ABE with proxy re-encryption [82] does not appear highly efficient for mobile users: the decryption process requires processing two subtrees instead of one, user revocation causes all user secret keys to be re-generated, and attribute revocation results in private key regeneration, too.

5.3 Proposed Algorithm

The proposed algorithm for key generation, distribution, and usage is now described. It consists of key management techniques that ensure highly secure data outsourcing to the cloud in a highly scalable manner for mobile cloud computing applications. Table 5.1 summarizes the symbolic notation used throughout the description. In the discussion that follows, improvements are proposed to the basic functions of the original CP-ABE scheme [17]; the security proof for the underlying cryptography appears in this related work and is relied upon here.

Symbol	**Description**
CSP	Cloud service provider.
M	Trusted manager.
T	Access tree structure.
R	Root node of T.
A	Set of attributes that must be satisfied against T.
U_o	Data owner.
U_r	Restricted user group.
m	Plaintext of user data.
CT	Ciphertext of user data.
v	Version of ciphertext.
PPK	Public partition key.
PSK	Secret partition key.
OPK	Owner public key.
OSK	Owner secret key.
DSK	Data secret key.
GPK	Public group key of U_r.
GSK	Private group key of U_r.
$DDSK$	Delegated data secret key.
$RK_{0\to x}$	Re-encryption key from version 0 to x.

Table 5.1: A legend for the symbolic notation used in the attribute-based model.

- A single authority does not generate all key material; the mobile data owner and cloud entity co-operate to jointly compute keys. The cloud provider has insufficient information to decode the user data that it permanently stores; yet, it assists in the distribution of a portion of the whole key material to all authorized users to minimize the communication cost for the data owner.
- The cloud has highly scalable computational ability, unlike a resource-constrained mobile user; a trusted manager also has greater computational resources than does a user. Pairing operations, which are the most expensive cryptographic operations that are involved in the proposed protocol, are thus performed by the cloud or manager to the maximum possible extent, relieving the burden on the mobile data owner.
- Proxy-based re-encryption has been integrated with CP-ABE so that the cloud provider may perform automatic data re-encryption; this is an optional feature that allows further control over revocation than is afforded by an attribute-based scheme alone, and it also takes advantage of the cloud provider's computational scalability. This dual-encryption scheme is a hybrid approach that offers greater flexibility in access control.

The proposed technique is now described as follows:

Preliminary:

Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be cyclic bilinear groups of prime order p with generator g. Also defined are random exponents $\alpha, \beta \in \mathbb{Z}_p^*$. The bilinear map e is a map such that: $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ with the following properties:

- Bilinearity: $\forall g_0, g_1 \in \mathrm{G}_0 : e(g_0^\alpha, g_1^\beta) = e(g_0, g_1)^{\alpha\beta}$.
- Non-degeneracy: $e(g_0, g_1) \neq 1$.
- Computability: $\forall g_0, g_1 \in \mathrm{G}_0$, there is an efficient algorithm to compute $e(g_0, g_1)$.

A secure one-way hash function $H : 0, 1^* \rightarrow \mathbb{G}_0$ is used as a random oracle and maps an attribute described as a binary string to a random group element.

Setup() $\rightarrow PPK, PSK, OSK$**:**

Suppose that Alice is a mobile user that acts as the self-elected data owner U_o of plaintext message m, which is user data that is desired to be encrypted and shared in

the cloud with other authorized users. If m exceeds the maximum allowed block length, then two solutions are possible: segmentation of the message may be performed, and the encryption applied to each individual segment; or, it is possible to first apply a symmetric cipher such as 256-bit AES to the entire message, then to encrypt the AES key itself using the proposed scheme, with the steps being reversed on decryption. Regardless, the length of the message does not impact the size or number of encryption keys required. In the case of message segmentation, the same pre-computed keys may be applied to all segments.

A manager M, acting as a trusted entity, chooses a random value α and computes g^{α} to form a private partition key PSK. It then performs a pairing operation to compute component $e(g,g)^{\alpha}$, which becomes one of the components forming the public partition key PPK. The public parameters $\mathbb{G}_0$ and g are also included in PPK. In the meantime, U_o chooses a secret data owner key OSK equal to β. It then computes the components g^{β} and $g^{\frac{1}{\beta}}$, the latter by first taking the inverse of OSK (i.e. $\frac{1}{\beta}$) in its possession; these components are added to the public key PPK, which is then uploaded and published in the public directory of the cloud. U_o does not divulge its secret OSK to any other party, including M. The elements of PPK, PSK, and OSK are as follows:

$$PPK = \left\{\mathbb{G}_0, g, g^{\beta}, g^{\frac{1}{\beta}}, e(g,g)^{\alpha}\right\}$$
$$PSK = \{\alpha, g^{\alpha}\}, \quad OSK = \{\beta\}$$

To provide an additional layer of security, the manager may create a shared secret group key GSK for an individual user or a restricted subset of users U_r, equal to a random value $u_0 \in \mathbb{Z}_p^*$, and a public key GPK as follows:

$$GPK = \{g^{u_0}\}, GSK = \{u_0\}$$

This group key is uploaded to the public directory as well. The secret key is not shared with the cloud; it may, however, be shared with the manager for the purpose of distribution to all authorized users. The initial version number of the secret key is initially referred to as 0, and will increase monotonically.

Encrypt$(PPK, GPK, m, T) \rightarrow CT$**:**

Any user may access the public partition key PPK, by downloading it from the public directory in the cloud, to perform an encryption; it need not necessarily be the data owner.

The encryption algorithm takes as input the key PPK and encrypts a message m under the tree access structure T with root R as described in [17]. It chooses a polynomial q_x for each node x in T, and a random value $s \in \mathbb{Z}_p^*$ that is applied to the PPK parameters. It sets $q_R(0) = s$ for the root node R, while Y denotes the set of leaf nodes in T. The function $\gamma(y)$ extracts the binary attribute string from a leaf node y in Y.

In order to protect highly sensitive data, the encryptor may wish to restrict user membership requirements beyond possession of the required attributes A. To do so, an additional key component may be incorporated consisting of $e(g,g)^{u_0 s}$, computed from g^{u_0}, which is the public key GPK of a restricted user group U_r belonging to the entire population of users. The group consists of one or more members, and the GPK is available from the public directory in the cloud. The absence of this component, where u_0 is presumed to be nil, will allow decryption based on satisfaction of the access tree only. The pairing operation $e(g,g)^{u_0}$ may be performed by the cloud or manager to assist the data owner.

The intermediate ciphertext CT_{own} is constructed as follows by the data owner U_o and uploaded to the cloud:

$$\begin{aligned} CT_{own} = \big\{ & v = 0, T, C_{0_{msg}} = m \cdot e(g,g)^{\alpha s}, C_{0_{grp}} = g^{u_0 s}, C' = g^{\beta s}, \\ & \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\gamma(y))^{q_y(0)} \big\} \end{aligned} \tag{5.1}$$

Next, the CSP performs a pairing operation on the $C_{0_{grp}}$ component in CT_{own} to obtain the result $\hat{C}_{0_{grp}} = e(g,g)^{u_0 s}$. The final ciphertext CT_0, denoting the initial version v of 0, is constructed as follows and published in the permanent data store of the cloud:

$$\begin{aligned} \tilde{C}_0 &= C_{0_{msg}} \cdot \hat{C}_{0_{grp}} = m \cdot e(g,g)^{\alpha s} \cdot e(g,g)^{u_0 s} = m \cdot e(g,g)^{\alpha s + u_0 s} \\ CT_0 &= \left\{ v = 0, T, \tilde{C}_0, C_{0_{msg}}, C' = g^{\beta s}, \forall y \in Y : C_y, C'_y \right\} \end{aligned} \tag{5.2}$$

Re-Encrypt$(CT_0, RK_{0 \to x}) \to CT_x$**:**

Whenever a user leaves the authorized membership of U_r, the user's access rights to the ciphertext must be revoked. When this occurs, a new version of the secret group key GSK is normally distributed by the manager to the remaining authorized users in U_r immediately, or distributed to each user on-demand through a secure off-line channel whenever data access is required. The CSP is then requested to perform a re-encryption operation on-demand so that its stored ciphertext can no longer be decoded using the prior

version of the key. The ciphertext is re-encrypted from version 0 to version x, given a re-encryption key $RK_{0\to x}$ from a user holding secret group key GSK_x assigned to version x; or, $RK_{0\to x}$ may be transmitted by the manager which is entrusted with the safekeeping of the key GSK_x. The re-encryption key is computed from the secret group key values u_0 and u_x corresponding to versions 0 and x of the ciphertext:

$$RK_{0\to x} = \left\{ g^{\frac{u_x}{u_0}} \right\}$$

The cloud provider computes the new ciphertext CT_x corresponding to version x (that is newer than the original version 0 uploaded by the encryptor), as follows, utilizing the component $C_{0_{msg}}$ found in CT_0 in Equation 5.2:

$$\tilde{C}_x = C_{0_{msg}} \cdot e(C_{0_{grp}}, RK_{0\to x}) = m \cdot e(g,g)^{\alpha s} \cdot e(g^{u_0 s}, g^{\frac{u_x}{u_0}}) = m \cdot e(g,g)^{\alpha s + u_x s}$$
$$CT_x = \left\{ v = x, T, \tilde{C}_x, C_{0_{msg}}, C_{x_{base}} = g^{u_x s}, C', \forall y \in Y : C_y, C'_y \right\}$$

The CSP is unable to decode the ciphertext during the re-encryption process as it has no knowledge of the old key u_0 and the new key u_x. The cloud provider retains the components $C_{0_{msg}}$ and $C_{x_{base}}$ in the ciphertext CT_x so that it may perform a future re-encryption from version x to y, where $y > x$.

KeyGen$(PPK, PSK, A) \to DSK$**:**

Irrespective of which party performed the encryption, the manager executes a data secret key generation algorithm which takes as input the private key PSK and a set of attributes A that are deemed sufficient to decrypt the ciphertext. Specifically, the manager chooses a random $r \in \mathbb{Z}_p^*$ and computes $(\alpha + r)$; it then exponentiates the component $g^{\frac{1}{\beta}}$ in the PPK by this sum to obtain the result $g^{\frac{(\alpha+r)}{\beta}} = (g^{\frac{1}{\beta}})^{\alpha+r}$. In this way, during the collaboration, the manager and data owner do not need to reveal their private keys PSK and OSK to one another. The data owner is not involved in the key generation and need not remain available.

To generate the additional required sub-parts of the data key, the manager chooses random $r_j \in \mathbb{Z}_p$ for each attribute in A. It computes the data secret key DSK that identifies with the attributes A as follows:

$$DSK = \left(D = g^{\frac{(\alpha+r)}{\beta}}, \forall j \in A : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j} \right) \tag{5.3}$$

The manager distributes a DSK based on a unique r value to each authorized user holding the required attributes A, without requiring the participation of the data owner. The manager may also provide the DSK to the data owner for peer-to-peer distribution at its discretion to the intended recipients of the encrypted message.

Decrypt$(CT, DSK, PPK, GSK) \rightarrow m$**:**

Any user that is authorized, by virtue of holding the required attributes A, may download the ciphertext CT from the cloud and decrypt it, as the recipient. The decryption routine takes as input the ciphertext CT and data secret key DSK obtained earlier either from the manager M or data owner U_o. The recursive decryption algorithm DecryptNode is applied to the root node R of the tree T that is publicly available on the cloud for download. If the node x is a leaf node, then let $i = \gamma(x)$, where the function γ denotes the attribute associated with the node x in T. If $i \in A$, then the DecryptNode function is defined as follows, using components D_i and D'_i derived from the DSK, as found in Equation 5.3, and C_x and C'_x derived from CT_{own}, as found in Equation 5.1:

$$\begin{aligned}\textsc{DecryptNode}(CT_{own}, DSK, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} = \frac{e(g^r \cdot H(i)^{r_i}, g^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} \\ &= \frac{e(g^r \cdot g^{\delta r_i}, g^{q_x(0)})}{e(g^{r_i}, g^{\delta q_x(0)})} = e(g,g)^{rq_x(0)}\end{aligned}$$

The recursive case, when x is a non-leaf node, is described in detail in [17]. If the access tree is satisfied by attributes A (that determined the data secret key DSK), observe that the DecryptNode function gives the following result:

$$\textsc{DecryptNode}(CT_{own}, DSK, R) = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$$

If the ciphertext is optionally encoded with the public key of the restricted user group U_r, then the recipient may utilize the secret key $GSK = u_x$ in conjunction with the $g^{\frac{1}{\beta}}$ component in the PPK to compute the required decryption component $g^{\frac{u_x}{\beta}}$.

The message m can then be decrypted as follows, assuming one round of re-encryption:

$$m = \frac{\tilde{C}_x \cdot e(g,g)^{rs}}{e(g^{\beta s}, D) \cdot e(g^{\beta s}, g^{\frac{u_x}{\beta}})} = \frac{m \cdot e(g,g)^{\alpha s + u_x s} \cdot e(g,g)^{rs}}{e(g^{\beta s}, g^{\frac{(\alpha+r)}{\beta}}) \cdot e(g^{\beta s}, g^{\frac{u_x}{\beta}})} = \frac{m \cdot \cancel{e(g,g)^{(\alpha+r+u_x)s}}}{\cancel{e(g,g)^{(\alpha+r+u_x)s}}}$$

Note that the component $e(g^{\beta s}, g^{\frac{(\alpha+r)}{\beta}})$ in the above equation is pre-computed by the manager, so that the user must perform only one pairing operation on decryption.

A summary of the key material in possession within the system is given in Table 5.2. The cryptographic operations described in this section, applied to a typical encryption and decryption transaction, are summarized in Table 5.3.

Entity	**Key material**
Data owner (U_o)	Chooses random β. Computes: $OSK = \{\beta\}$, $OPK = \{g^{\beta}, g^{\frac{1}{\beta}}\}$ Chooses random u_o. Computes: $GSK = \{u_0\}$, $GPK = \{g^{u_0}\}$ Shares GSK with user B. Chooses random s. Computes CT_{own} based on the PPK and GPK, and uploads it to CSP.
Manager (M)	Chooses random α. Computes: $PSK = \{\alpha, g^{\alpha}\}$, $PPK = \left\{\mathbb{G}_0, g, g^{\beta}, g^{\frac{1}{\beta}}, e(g,g)^{\alpha}\right\}$ Chooses random r. Computes: $D = g^{\frac{(\alpha+r)}{\beta}}$, and DSK based on attributes A. Distributes DSK to user B.
Cloud (CSP)	Publishes PPK and GPK in a public directory. Optionally performs a pairing and computes CT_0 from CT_{own} and GPK. Stores CT_0 in the permanent data store.
Recipient (B)	Downloads CT_0 and decrypts it using the DSK and GSK.

Table 5.2: A summary of the key material in the attribute-based model.

5.4 Optional Features

The original CP-ABE scheme defined delegation, where following the generation of the data secret key DSK, the manager may choose to delegate access to a user possessing a particular subset of the required attributes. For completeness, the integration of this optional operation with the proposed technique is shown:

Delegate$(DSK, S') \rightarrow DDSK$**:**

The delegation algorithm first takes as input a data secret key for a set of attributes A, and a subset $A' \subseteq A$. The manager chooses a new random $r' \in \mathbb{Z}_p^*$ and computes the result:

$$D' = D \cdot (g^{\frac{1}{\beta}})^{r'}$$

The manager then chooses a new random r'_k for each attribute k in the subset S' to form the next sub-part, and creates a new secret delegation data key $DDSK$ for S':

$$DDSK = \left(D' = D \cdot (g^{\frac{1}{\beta}})^{r'}, \forall k \in A' : D'_k = D_k \cdot g^{r'} \cdot H(k)^{r'_k}, D''_k = D'_k \cdot g^{r_k} \right)$$

Since the delegation key is randomized with the value r', it is equivalent in its level of security to the original key DSK. Note that the data owner is not involved in this operation.

5.5 Discussion

The proposed scheme offers a dual layer of security through attribute-based encryption and also public key encryption which may be optionally applied. If the secret group key GSK is compromised, the data is still safeguarded; only the users that have the required attributes will be able to decrypt it. The interception of any key components over the network, including the re-encryption key, will not yield useful information to the attacker, as no private keys are transmitted in the clear. Furthermore, the algorithm achieves collusion resistance because the $e(g, g)^{\alpha s}$ term of the ciphertext cannot be recovered by an attacker even if the manager's or a user's private keys are compromised.

Any user may encrypt data using the public partition key PPK stored in the cloud. However, the cloud provider is unable to decrypt any user data stored on its premises as it cannot access the secret owner and data keys OSK and DSK. Nor is useful information revealed to the CSP during the re-encryption process. The encryptor of a message may restrict its eligible readership by not only selecting a required set of attributes, but also through the optional use of a public group key which may be shared by a group or simply possessed by a single user; the trade-off made is the required distribution of the group key and the extra pairing operation required during the encryption phase, but it is advantageously computed by the cloud provider. The manager can also assist with distribution.

Critically, the performance implications are modest for the mobile users. The data owner must only perform exponentiation operations during its key generation phases, while the manager performs the more expensive pairing operation during partition key generation in the SETUP algorithm. Also, with the assistance of the manager, the user performs only a single pairing operation on decryption.

	Alice (U_o)	**Cloud** (CSP)	**Manager** (M)	**Bob** ($\in U_r$)
1	Generates private owner key OSK and sends public component OPK to M to form partition key PPK. Generates private and public group keys GSK and GPK to share with trusted users and CSP, respectively.	Stores partition key PPK obtained from M in a public directory for dissemination to all authorized users. Also stores the public group key GPK that it obtains from Alice.	Generates private and public partition keys PSK and PPK, the latter with assistance from Alice, and uploads PPK to CSP.	Obtains GSK from Alice as a trusted user.
2	Assuming that Alice is also the encryptor, encrypts message m, with PPK and under tree T, as CT_{own}, and uploads it to CSP for storage. Also generates a component for data key DSK from OSK.	May assist the encryptor with generation of ciphertext CT_{own}. Computes CT_o from CT_{own} and GPK, and stores it as ciphertext version 0 in permanent storage, for dissemination to all authorized users.	Generates data key DSK from its private partition key PSK based on attributes A, with assistance from Alice. Distributes the DSK to all authorized users.	Obtains the DSK from Alice or M.
3				Downloads CT_o from CSP and decrypts it to yield the plaintext m.

Table 5.3: A summary of operations in attribute-based re-encryption, with participating user actors shown, and assuming no additional re-encryption occurs.

5.6 Implementation

The proposed protocol was implemented and profiled to gauge its performance. It was realized on popular existing commercial platforms, including the Google Android mobile and the Google App Engine cloud platforms. A simulation calibrated to the performance benchmarks was then run to examine the scalability of the proposed algorithm.

5.6.1 Performance Measurement

An existing implementation in Java [114] that relies upon the original CP-ABE scheme [17] served as the baseline implementation. From this starting point, the implementation was significantly rebuilt to reflect the proposed protocol described herein. The implementation uses the Java Pairing-Based Cryptography Library (jPBC) version 1.2.1 [33], a port of the PBC (Pairing-Based Cryptography Library) in C [76]. The use of Java 6 Standard Edition permits the protocol to be ported to a wide range of computing environments.

The implementation was run on different computing hosts to assess their relative performance. Refer to the implementation model in Figure 5.1. On the client end, a simulation was run on a desktop platform consisting of an Apple iMac with a quad-core 64-bit 3.4 GHz Intel Core i7 processor with 16 GB of RAM, running Mac OS X 10.8.2 (Mountain Lion). Additionally, it was run on an older Google Nexus One smartphone with a 1 GHz Qualcomm Scorpion processor with 512 MB memory running Android OS 2.3.6 (Gingerbread), and a new Samsung Galaxy Note II smartphone with a quad-core 1.6 GHz ARM Cortex-A9 processor with 2 GB of RAM, running Android OS 4.1.1 (Jelly Bean). On the server end, a lowest-class F1 front-end instance was run as a Java servlet application on the Google App Engine (GAE) cloud, configured at the equivalent of a 600 MHz processor with 128 MB of RAM. A connection was established between the desktop or mobile Android client and an instance running on the GAE cloud via HTTP requests, using JSON for data interchange and the Google Gson library for marshalling between Java objects (used by the Java client and server implementations) and the JSON representation. Note that the security model of GAE does not allow direct network connections and native code execution. Only a subset of the Java 2 Standard Edition (J2SE) SDK 1.6 classes are whitelisted on Android and GAE; fortunately, the required Java Cryptography Extension (JCE) classes are supported.

The simulation consisted of multiple iterations of encryption and decryption using the

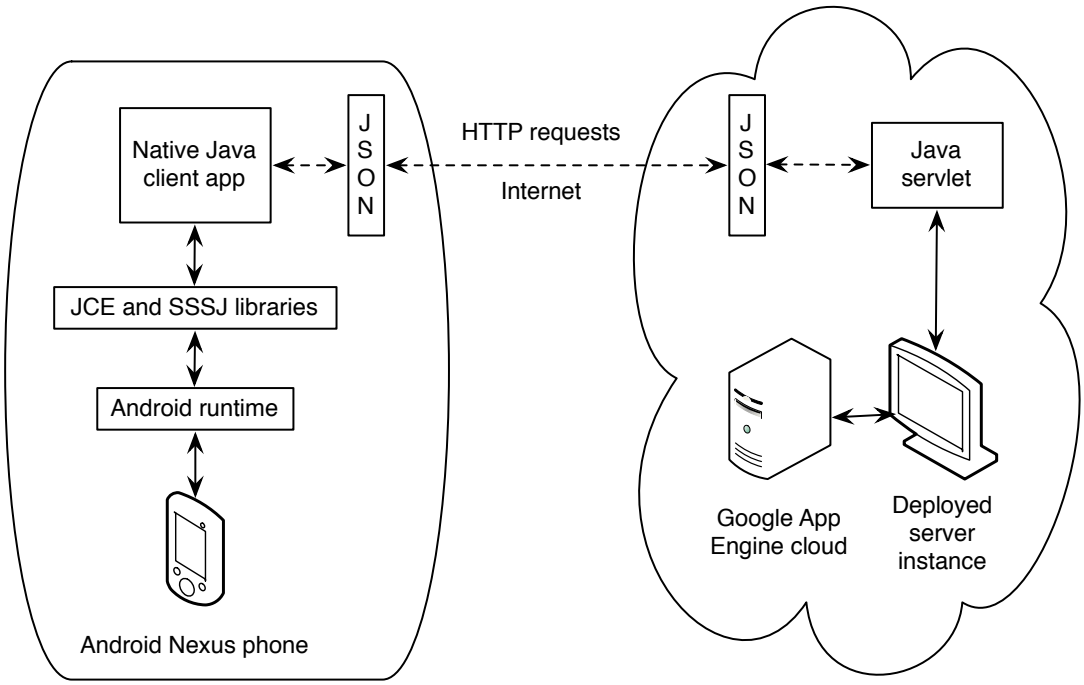


Figure 5.1: A high-level model of the implementation of attribute-based re-encryption.

proposed functions defined in Section 5.3, using a single-attribute policy to configure an environment that ran at the fastest possible speed. A *Type A pairing* was utilized in the algorithm with a group order size of 160 bits and a base field order size of 512 bits, which is the default curve configuration in the jPBC library test code, and is suitable for cryptographic use. Performance benchmark results are shown in Table 5.4, showing the average execution times of all main cryptographic operations calculated from simulation runs on the same iMac desktop computer platform to permit direct comparison. Simulations from the original BSW algorithm [17] as well as the proposed algorithms are shown; in the latter case, one set of runs was made using an optional group key to additionally secure the plaintext, with a round of re-encryption included, and one set of runs was made without the benefit of a group key, to permit comparison.

Next, operations were repeated on the appropriate platform (desktop, mobile, or cloud) for each operation, to ascertain realistic timings in a mobile cloud computing system. A user interface was built for the Android app to allow execution of all algorithms locally or on a Google App Engine instance in the cloud, as depicted in Figure 5.2. The results are summarized in Table 5.5, with the appropriate platform chosen to represent a typical device executing each operation in question. Note that the underlying jPBC library is not optimized for a constrained mobile operating environment; such optimizations may often yield very significant performance improvements in practice. For instance, careful memory allocation and maintenance of a small footprint may reduce the expensive garbage collection events observed in the Android device system logs during execution. The comparative benchmarking results are shown visually in Figure 5.3, with the following observations:

- In comparison to the baseline implementation, the key setup activity in the proposed protocol is approximately evenly split between the data owner and the manager, which is of high benefit given that the owner is presumed to be a resource-constrained mobile user.
- The key generation and encryption activities are approximately equal in terms of computational requirements. The utilization of a group key only applies an approximately 30% penalty to encryption, which is borne by the CSP because it is responsible for the pairing operation, not the data owner.
- Crucially, the data owner does not participate in the data secret key generation activity in the proposed protocol; the manager does so instead.

Algorithm	**Task**	**Timings (in ms)**				
		μ	σ	s	LB	UB
Baseline (BSW)	Owner setup	47.5	28.1	787.0	0	102.5
	Keygen	70.0	3.7	13.6	62.8	77.2
	Owner encryption	61.3	2.9	8.2	55.7	66.9
	Decryption	23.9	1.5	2.3	20.9	26.8
Proposed (no group key)	Owner setup	29.8	45.3	2,053.5	0	118.6
	Manager setup	19.8	2.1	4.2	15.7	23.8
	Keygen	74.6	5.4	28.7	64.1	85.1
	Owner encryption	65.0	4.9	24.2	55.3	74.6
	Decryption	24.5	2.5	6.1	19.6	29.3
Proposed (with group key)	Owner setup	37.6	26.7	710.4	0	89.8
	Manager setup	18.8	1.2	1.5	16.4	21.2
	Keygen	70.0	3.6	12.7	63.0	77.0
	Owner encryption	60.8	2.3	5.2	56.3	65.2
	Cloud encryption	18.6	1.0	1.0	16.6	20.5
	Decryption	42.8	2.1	4.6	38.6	47.0
	Reencryption setup	21.6	1.0	0.9	19.7	23.5
	Reencryption	7.5	0.6	0.4	6.2	8.8

Table 5.4: The performance results obtained from the attribute-based implementation running on an iMac desktop computer. The following symbols are used: μ is the sample mean, σ is the sample standard deviation, s is the sample variance, and LB and UB are the lower and upper bounds, respectively, of the 95% confidence interval, for each set of runs using each of the baseline and proposed algorithms. 100 runs were executed using each technique, and all operations were performed using a single-attribute policy. The data owner performed a pairing operation on encryption in all cases, without assistance, which accounts for its slower operation.

- Furthermore, the data owner is not required to perform costly pairing operations, including in the generation of ciphertext (although it does so in the implementation).
- The optional re-encryption operation is only a fraction, approximately 40%, of the total encryption operation in terms of the period of computation, and is also performed by the CSP without burdening the data owner; also, it has the potential to be scaled by allocating additional cloud instances as required.

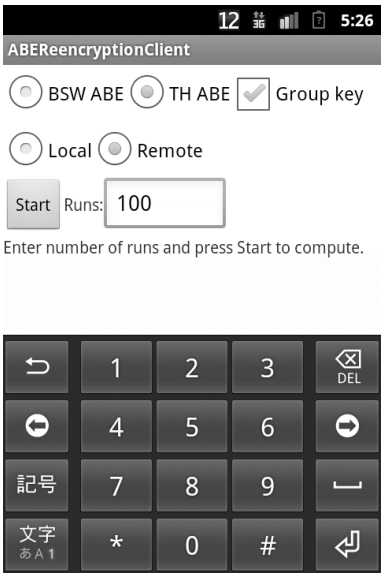


Figure 5.2: The user interface of the mobile client app in attribute-based re-encryption.

Cryptographic function	**Device**	**Baseline (BSW)**	**Proposed (no GK)**	**Proposed (with GK)**
SETUP by data owner.	Note II	1157	396	586
SETUP by manager.	GAE	n/a	278	219
KEYGEN.	GAE	791	749	786
ENCRYPT by data owner.	Note II	1273	1250	1239
ENCRYPT by CSP.	GAE	n/a	n/a	657
DECRYPT.	Note II	1364	1391	2043
RE-ENCRYPT setup.	GAE	n/a	n/a	247
RE-ENCRYPT.	GAE	n/a	n/a	130

Table 5.5: The performance benchmarks used for calibration of the attribute-based simulation. All timings are in ms, with "Note II" denoting the Galaxy Note II mobile phone, and "GAE" denoting an F1 instance running on a GAE cloud servlet.

5.6.2 Simulation

A custom simulation program was developed that permits an assessment of the scalability potential of the proposed scheme. The simulation program was executed on a desktop computer but was calibrated with the function timing results from the benchmarks obtained as described in the previous section; that is, the timings served as the basis for calculating the accumulated processing workload of the various entities in the system as a result of performing the various defined cryptographic operations in response to simulated user actions. Various parameters may be adjusted in the simulation to highlight the differences in the algorithms discussed, including the original CP-ABE and the proposed schemes.

An initial unauthorized user population is modelled, and in each round of the simulation, users randomly join or leave a user set that is authorized to access a particular data record. A single data owner responsible for data encryption is modelled. Each user randomly takes an action each round, with some predefined probability; actions include accessing the encrypted data and performing a decryption, or joining or leaving the authorized user set and thus triggering appropriate key generation activities. The encrypted data record stored in the cloud may also be replaced in a round by the data owner, once it has outlived its usefulness, with more recent data; this initiates a new key setup phase.

In Figures 5.4, 5.5, 5.6, and 5.7, the simulation results for one typical simulation run of each algorithm are shown, with the processing workload shown over time for each entity (the data owner, the manager, the CSP, and the total set of users involved in accessing the data record stored in the cloud). The workloads are directly based on the cryptographic function profiling results found in Section 5.6.1 so that calibration was done with real-world data obtained from the practical implementation. The simulation was run with values for adjustable parameters as specified in Table 5.6. The irregularities found in the plots are due to the probabilistic nature of the events executed in one sample simulation execution.

The following observations may be made with respect to the results of the illustrated runs showing various dominant roles in the system:

1. In the original BSW algorithm [17], the dominant workload is undertaken by the data owner, which participates in not only the encryption of the user data, but also in the data secret key generation for each new user, as shown in Figure 5.4. The owner must also re-generate keys for all users whenever a revocation occurs, without

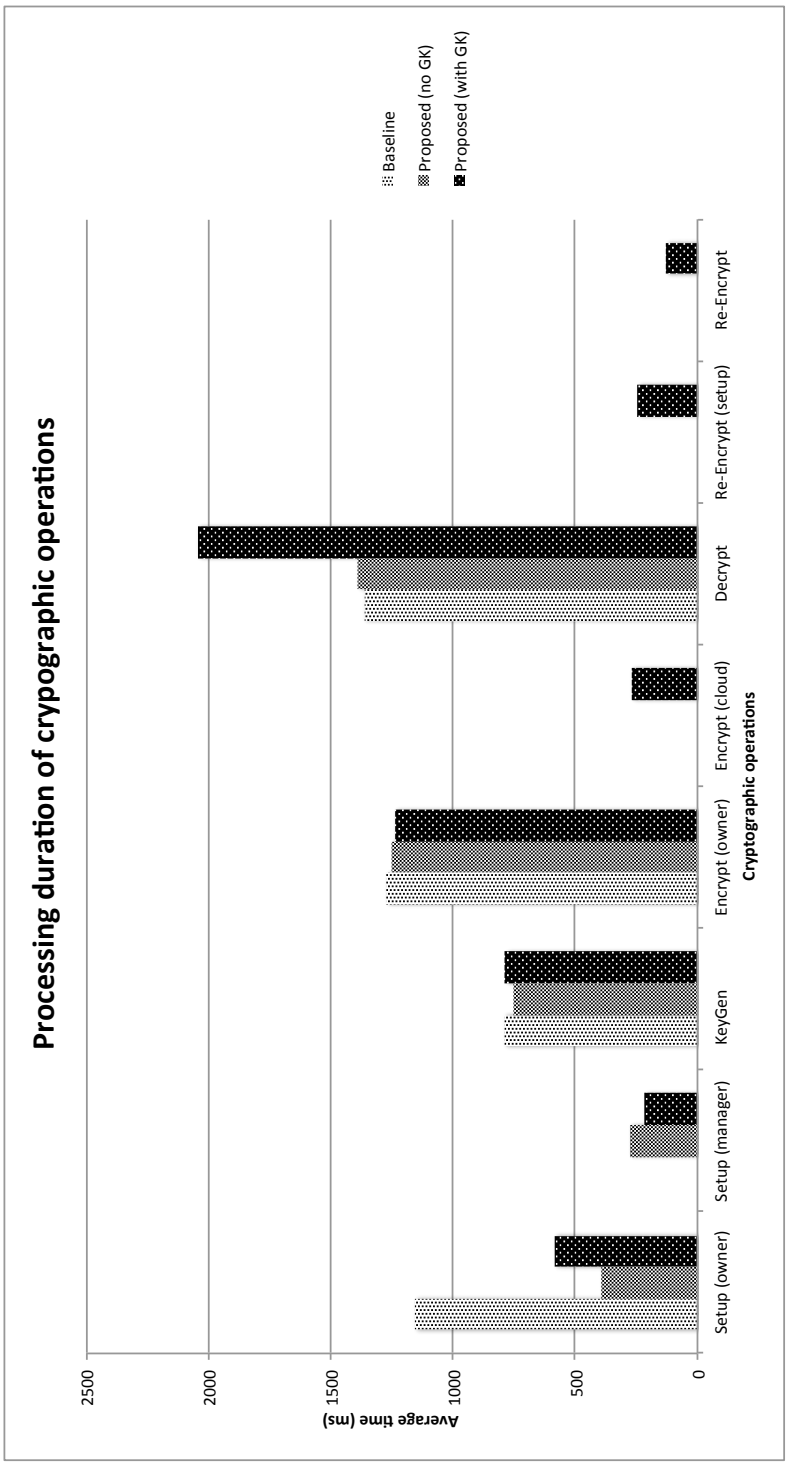


Figure 5.3: The performance results obtained from the attribute-based implementation, showing the processing time of cryptographic operations in the baseline and proposed protocols.

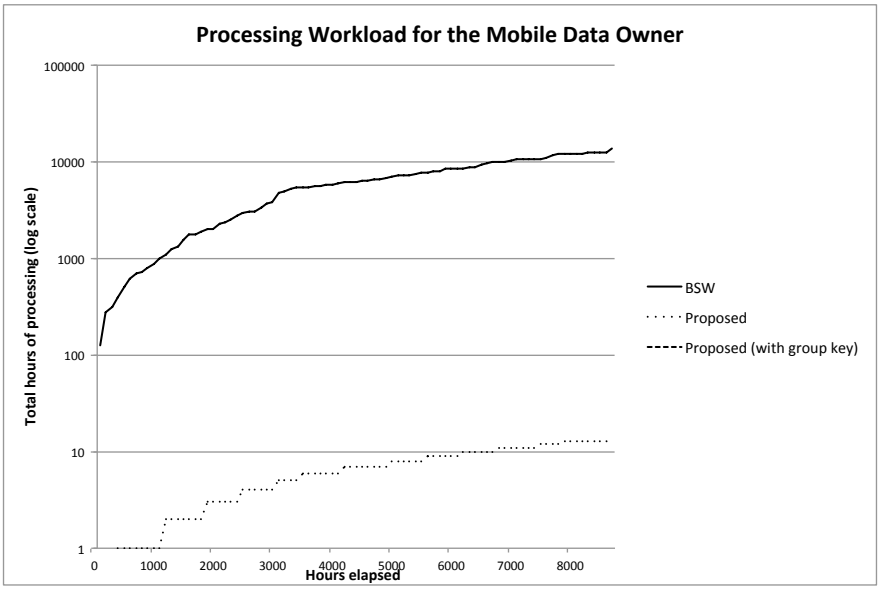


Figure 5.4: The processing workload for the data owner in the attribute-based model.

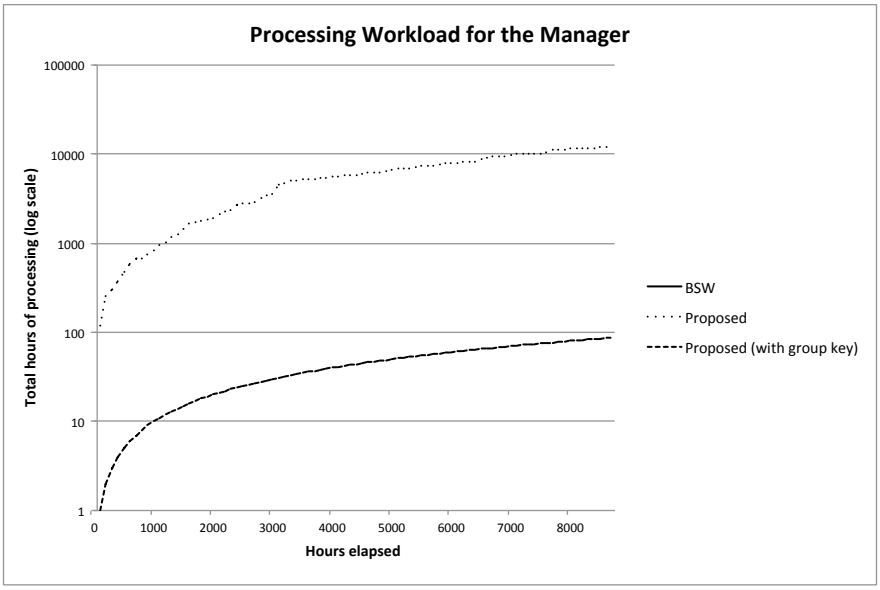


Figure 5.5: The processing workload for the manager in the attribute-based model.

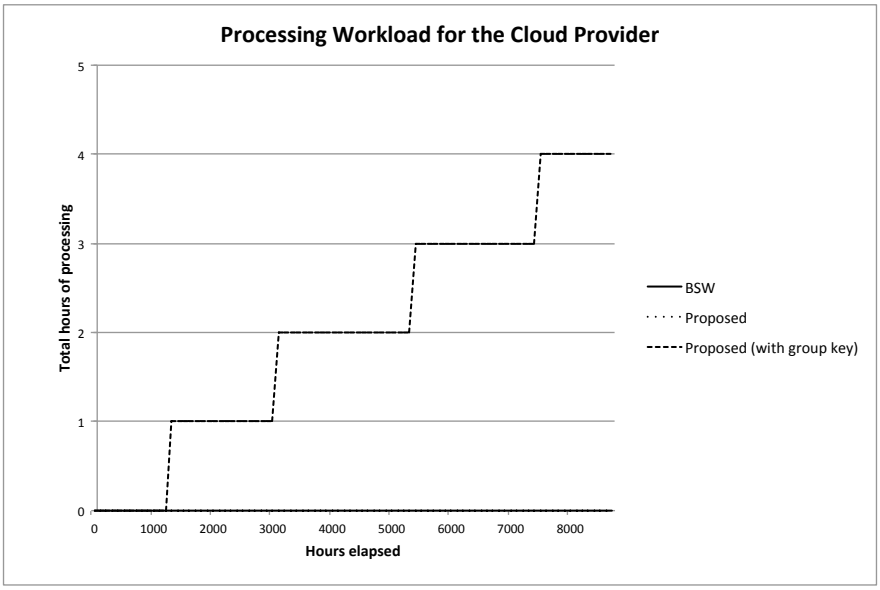


Figure 5.6: The processing workload for the cloud provider in the attribute-based model.

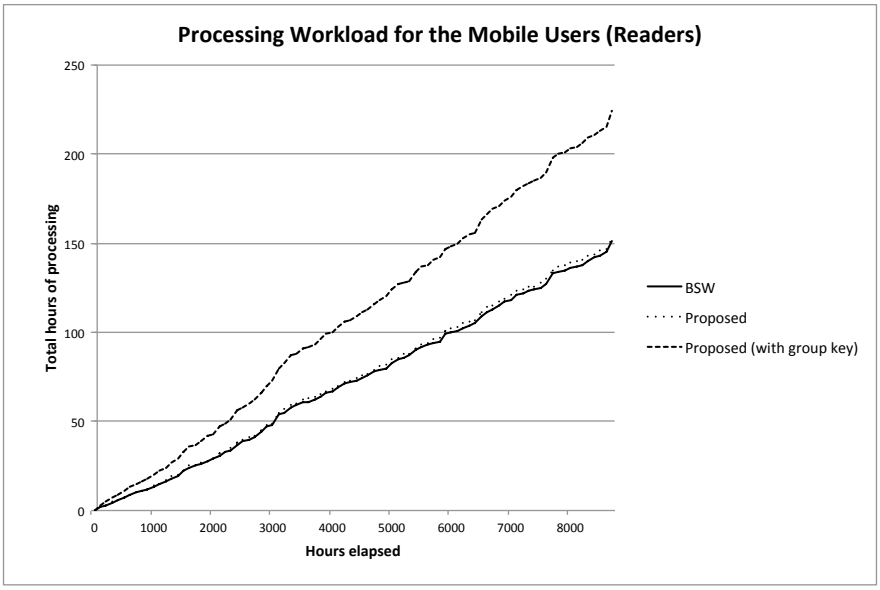


Figure 5.7: The processing workload for the user population in the attribute-based model.

Parameter	**Value**
Initial unauthorized user population	10,000 users
Length of each round	1 hour
Total length of all rounds simulated	1 year
Probability of a user joining the authorized set	0.5%
Probability of a user leaving the authorized set	0.5%
Probability of a user downloading the cloud data	5%
Probability of the cloud data being replaced	5%
Total joins in simulation run	397,000
Total accesses in simulation run	396,000
Total leaves in simulation run	40,000
Total data replacements in simulation run	419

Table 5.6: The parameters used for the attribute-based simulation.

assistance of any other network entity, based on the assumption that revocation is only possible through modification of attributes for the user in question. Since the data owner is presumed to be a mobile device in the assumed system model, the scalability potential appears inadequate.

2. In the proposed algorithm without the use of a group key, the manager becomes responsible for the main workload of the key re-generation activity, which entails a pairing operation, as shown in Figure 5.5. The manager is expected to be able to scale accordingly to meet the processing demands, but requires sufficient client infrastructure to do so, which may be uneconomical.

3. In the proposed algorithm with the use of a group key, the manager is still responsible for most of the key re-generation activity, but revocation is now handled through re-encryption of the group key, a task performed mainly by the cloud provider, as shown in Figure 5.6; this results in a much lower overall workload in the system. The additional decryption cost for the reader population is significant but acceptable, as shown in Figure 5.7. In practice, a decryption will only occur if data has been modified and has to be fetched from the cloud again. Hence, this algorithm is considered the best candidate for a highly scalable mobile cloud computing application within the system model described.

5.7 Summary

A key management system has been proposed for secure data outsourcing applications, whereby attribute-based encryption effectively permits authorized users to access secure content in the cloud based on the satisfaction of an attribute-based policy. The scheme has been modified so that a data owner and a trusted authority co-operate in the key generation and encryption processes such that computationally-intensive cryptographic operations and requests are minimized for the data owner; this is of importance to a population of mobile users that must conserve their consumption of battery and usage of wireless communication. In particular, the user is not required to perform costly pairing operations; instead, they are delegated to the manager and cloud provider. Also, the manager computes the decryption key, not the data owner, and it assists with key distribution on behalf of the owner.

Furthermore, a hybrid protocol is proposed that optionally allows message encryption based on a group key, allowing the user membership to be further refined for highly sensitive data. Additionally, it allows re-encryption to occur, and thus revocation to become efficient without necessitating existing common remedies and their limitations; an example is the expiration of attributes specified in the attribute-based policy that leads to constant key updates as time elapses. The proposed protocol is similar in overall performance to the original ciphertext-policy attribute-based-encryption idea, while significantly lessening the computational and traffic burden on the mobile data owner in a system where data updates and encryption activities are frequent and dominant. Thus, the proposal is useful for securing mobile cloud computing with very large user populations.

Chapter 6

Cloud-Hosted Key Sharing

6.1 Introduction

The key management schemes proposed thus far have involved performing computationally intensive key re-generation operations within the cloud to take advantage of its scalability; these computations, however, may prove too costly in certain cloud systems where such processing overhead cannot be justified. This chapter suggests utilizing the principle of key sharing and thus concentrating on the utility of another highly economical asset of a cloud system: its permanent replicated storage, which can scale according to client demand, and is typically billed at a small fraction of a dollar per GB of data per month [6]. The key design factors for a cloud-based secure storage system that motivate this chapter include: no additional server-side logic being required on the cloud provider end to support cryptographic functions; fine-grained data access; highly scalable sharing among multiple readers and writers; minimal computation required by mobile users; minimal communication required with the cloud provider; and no inherent trust of the provider existing, in terms of the administrator having unrestricted access to stored user data. The system model described in Section 2.2.2 on page 14 and the adversary and threat model described in Section 2.4.1 on page 19 are assumed.

In this chapter, related work on key sharing is presented in Section 6.2. Next, a key management scheme based on cloud-hosted key sharing is proposed in Section 6.3. Options applicable to various additional use cases and cloud variants are presented in Section 6.4. A

practical implementation with benchmarking results, as well as an additional simulation to assess scalability, are discussed in Section 6.5. Concluding remarks are made in Section 6.6.

The content of this chapter is based on work that has been published [110].

6.2 Related Work on Key Sharing

NIST, in its Electronic Authentication Guideline [21], recommends secret sharing to protect long-term credentials in its level 3 security definition for a CSP. Secret key sharing allows a secret such as key information to be divided into multiple shares [98]; these shares may be distributed among key generators using the concept of threshold decryption [19], or portions of a private key are distributed among users [10]. The challenge is that the client must assemble a key from multiple sources, potentially resulting in expensive communication overhead. Rather than key shares being distributed on-demand by some authority, it has been proposed that they be distributed across a network of nodes whose accessibility is subject to degradation over time. The Vanish system [41] distributes shares onto a DHT (Distributed Hash Table) that underlies a peer-to-peer file sharing network. It suggests the concept of *self-destructing data*, where copies of data become unreadable over time due to the effect of user churn on the index. The problem with adapting the scheme to a cloud-based context is that it relies upon the availability of the shares among the nodes, which cannot be guaranteed. It requires that each user obtain key shares from multiple other nodes that form the index, which is an expensive proposition if the user is operating a mobile device. In the DepSky storage system [16], shares are necessarily distributed across multiple clouds to form distributed trust and to restrict access. Each cloud provider has access to a single share and thus cannot decode the stored data; this requires support for a cloud-of-clouds. Also, because the data shares are unencrypted, each cloud must be independent and collusion assumed to be impossible. SafeVanish is a system for self-destructing data in the cloud that prevents a sybil attack to which the original Vanish system is vulnerable [121]; however, it is still reliant upon a DHT implementation; whether the DHT is publicly or privately hosted, the process of key destruction is never under the direct control of the client in such a system, thus limiting its flexibility.

6.3 Proposed Algorithm

The following algorithm is based on the principle of limiting access to encrypted data in the cloud through the process of storing and removing encrypted key shares in the cloud. Additional variants are also presented, such as the usage of a cloud-of-clouds to distribute the key shares, and a trusted manager to perform key rotation and remove this burden from the data owner. Table 6.1 provides the meaning of symbols used in the discussion.

6.3.1 Main Technique

Key Generation and Encryption

Consider a technique based on Shamir's secret sharing [98]. U is the set of users accessing the cloud, and an access structure Γ_U is a list of subsets of U such that each subset is trusted. Any trusted subset U_{tr} of parties, where $U_{tr} \in \Gamma_U$, can recover the secret from the set KS of shares stored in the cloud. Any untrusted subset, however, cannot obtain information about the secret. The access control structure can be defined such that any $(t+1)$ or more parties in U can recover the secret, while any t or less cannot do so; this secret sharing scheme is *threshold-based* (as defined in [98]).

Refer to Algorithm 6.3.1. In the ENCRYPT operation, Alice, or user A, proceeds to generate key shares and encrypt a message m to be stored in the cloud and identified with a unique identifier m_{id}. User A generates a symmetric key K (such as an AES, or Advanced Encryption Standard, key) and divides it into multiple shares $KS[1]$ to $KS[n]$, where n is the current total number of shares; a minimum of $t+1$ shares are required for decryption, where $t+1 \leq n$. Parameter t may be decreased or increased in value to adjust the level of security, while parameter n determines the number of users supported and the storage requirements for the shares. Each share $KS[i]$ is encrypted as $EKS[i]$, using a symmetric encryption key $AK[i]$ (such as an AES key) belonging to user A, known as an access key; it is also possible for the same access key $AK[i]$ to protect multiple shares to conserve storage and communication costs. The encrypted shares are stored in a public key database in the cloud and cannot be read in plaintext form by the provider, although they remain accessible for download by users. A requests sufficient storage in the cloud to hold n shares, which represents an upper bound; as described later, a replacement strategy will

Symbol	**Description**
U_{tr}	Authorized user set.
M	Trusted manager.
A, B, C	Users Alice, Bob, and Charlie in U_{tr}.
$GenKey()$	Function to generate a random key of some predetermined length.
K	Symmetric data key.
$KS[i]$	Share i of key K.
$v[i]$	Version associated with a key share $KS[i]$.
$EKS[i]$	Encrypted key share i.
$EKS[i]_{hdr}$	Metadata header for key share $KS[i]$.
$Partition(K, i, n)$	Function to generate share i of key K, where n is the total number of shares.
$EncryptSym_y(x)$	Function to encrypt data x using symmetric key y (such as AES).
$DecryptSym_y(x)$	Function to decrypt data x using symmetric key y (such as AES).
$Hash(x)$	Compute the digest of message x.
$Reconstruct([z])$	Function to reconstruct secret key from shares in array $z[]$.
$AK[i]$	Access key to unlock share $KS[i]$.
m	Plaintext of user data.
m_{id}	Unique plaintext record identifier.
c	Ciphertext of user data.
t	The threshold number of key shares KS, above which (at $t+1$ or greater), there is a sufficient number to compute K.
n	The total number of key shares KS generated for a particular data record.
L	A description key identifying the set of key shares eligible to decrypt c.
PK_X	Public key of user X.
SK_X	Private (secret) key of user X.

Table 6.1: A legend for the symbolic notation used in the key-sharing model.

replace older shares with newer ones while utilizing the same total capacity. The message flow is summarized in Table 6.2, encompassing the usage of a single version of a data key.

A high-level data flow diagram appears in Figure 6.1 for reference in the discussion that follows. As will be elaborated, a mobile data owner is responsible for generating and uploading content intended for encrypted storage in the cloud; it also creates key material that can be used to decrypt the cloud data, and this material is also securely deployed in the cloud. Multiple mobile readers of the same content exist, and access portions of the key material to carry out read operations on the cloud data that they download. Concurrently, a cloud application manages access to the stored user data and performs required maintenance activities on the key material that it holds on behalf of users. A trusted intermediary is not required in the basic system model.

Algorithm 6.3.1: ENCRYPT(m, m_{id})

comment: Generate the encrypted data and access keys.

$K \leftarrow GenKey()$

for $i \leftarrow 1$ **to** n

do $\begin{cases} KS[i] \leftarrow Partition(K, i, n) \\ AK[i] \leftarrow GenKey() \\ EKS[i] = EncryptSym_{AK[i]}(KS[i]) \\ EKS[i]_{hdr} = \{m_{id} \parallel i \parallel v[i]\} \end{cases}$

comment: Encrypt the plaintext message.

$c \leftarrow EncryptSym_K(m)$

The plaintext user data m requiring protection is assigned a unique record identifier of m_{id} and encrypted by A as ciphertext c using K; it is uploaded to the provider and is stored in the cloud. Since the cloud provider cannot unlock any share stored in the key database, it is unable to decode c. To the ciphertext of the user data is appended a description key L identifying the set of key shares eligible to decrypt the data, of which only the threshold amount is required by any user.

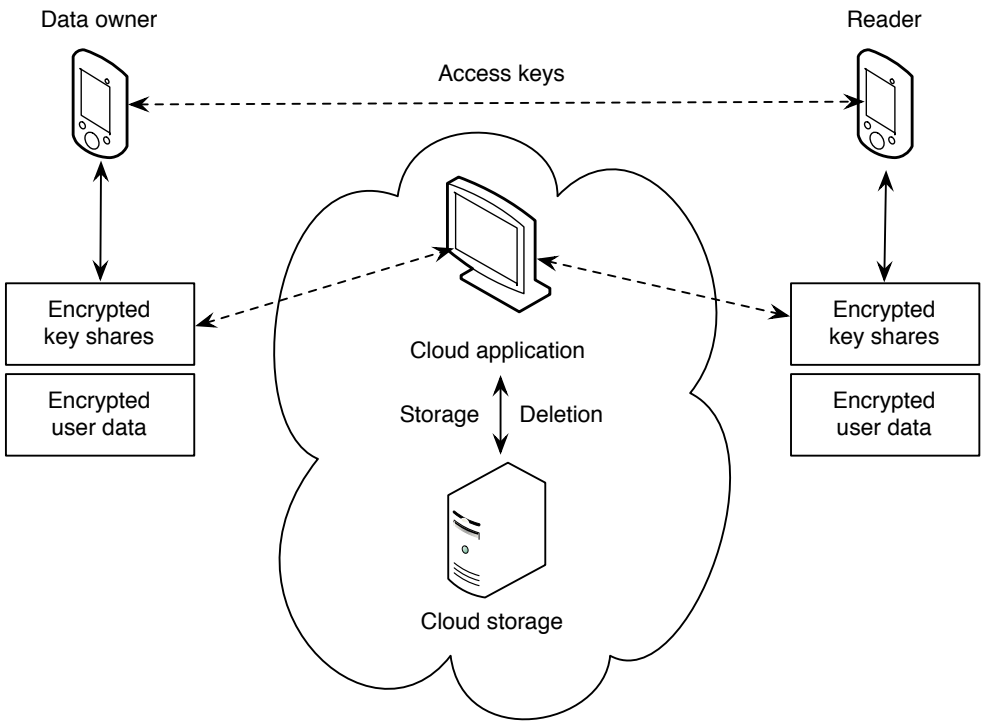


Figure 6.1: The high-level data flow in the key-sharing system.

A	**Cloud**	B
ENCRYPT(m)	$\underline{[c, L, EKS[0..n]]}\rightarrow$	Store $c, L, EKS[0..n]$.
	$\leftarrow\underline{\quad AK[x] \text{ to } AK[y] \quad}\rightarrow$	
	(where $y - x \geq t$)	
	$\leftarrow\underline{[c, L, EKS[x..y]]}$	DECRYPT(c)
	Periodically erase shares from 0 to n.	
	$\leftarrow\underline{\quad \text{Optionally request additional shares to satisfy threshold } t+1. \quad}\rightarrow$	DECRYPT(c)

Table 6.2: The message flow in the key-sharing model.

Metadata

A will create a metadata header $EKS[i]_{hdr}$ for each share i and upload it with the encrypted share payload. The metadata will consist of the following fields: the record identifier m_{id}, the key share identifier $i \in \{1..n\}$, and the key share version v.

The assumed adversary model does not require integrity controls, as the cloud provider is regarded as being honest-but-curious. However, if this assumption does not hold true, and verifiability of the downloaded shares is required, then a digital signature of the key share may be created. A mechanism for utilizing the header as a signature is proposed in [110], such that tampering of c is detected through a comparison of message digests; the signing operation requires a supporting public key infrastructure such as RSA.

Decryption

Refer to Algorithm 6.3.2. Bob, or user B, wishes to access c, and so he executes the DECRYPT operation. Suppose that B is an authorized member of U_{tr}. B obtains symmetric access keys $AK[x]$ to $AK[y]$ from A, where the range of keys is of at least size $t+1$, the required threshold; this assumes that each key $AK[i]$ permits decryption of the key share $KS[i]$ stored in the cloud, where i is in the range 1 to n, such that all shares are in L. Again, it is also possible to have one access key unlock multiple shares, instead. Regardless, every user is given a random set of access keys in the initial allocation; each set satisfies the threshold at a minimum.

Algorithm 6.3.2: DECRYPT(c, m_{id})

comment: Reconstruct the data key.

for $i \leftarrow 1$ **to** $t+1$
  do $\{ KS[i] \leftarrow DecryptSym_{AK[i]} EKS[i]$

$K \leftarrow Reconstruct(KS[1,..,(t+1)])$

comment: Decrypt the plaintext message.

$m \leftarrow DecryptSym_K(\mathrm{c})$

For instance, in one sample configuration, suppose that the total number of shares n for a particular data record is 100, and that the number of shares required for decryption, $t + 1$, is three. Refer to Figure 6.2. Data owner A will provide access key $AK[1]$ to B, which provides access to five shares stored in the cloud; only three are required. B may download the ciphertext, as well as all three encrypted key shares, directly from the cloud and within the same request. B will then be able to decrypt the required user data.

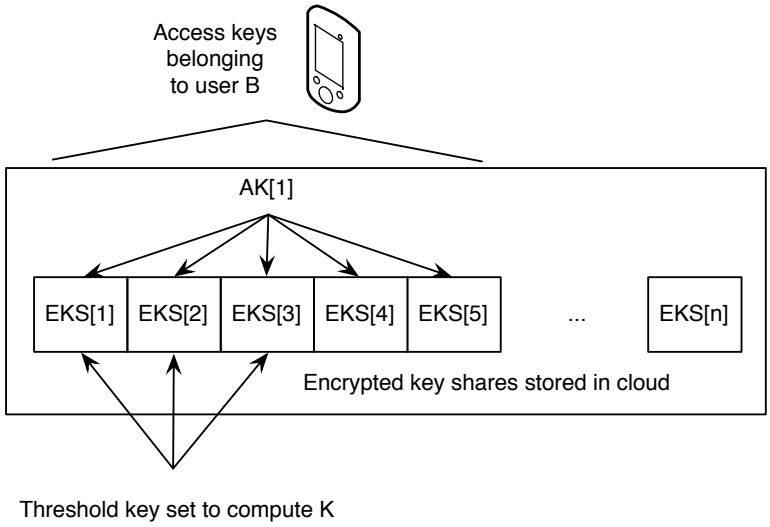


Figure 6.2: An example of key assignment in the key-sharing model.

The entire user population may be significantly greater in number than the total number of shares n; thus, the same key shares may be randomly assigned to multiple users. For instance, Charlie, user C, is also an authorized member of U_{tr} and is allowed to access the ciphertext c. Owner A may issue the same access key $AK[1]$ to C, to unlock the same key shares in common with user B. Note that if an access key unlocks only a single key share, instead, then finer-grained control is attained; in that case, user B can be given access keys $AK[1]$ to $AK[5]$ to unlock shares $EKS[1]$ to $EKS[5]$, and user C can be given access keys $AK[3]$ to $AK[7]$ to unlock partially overlapping shares $EKS[3]$ to $EKS[7]$. However, this flexibility is at the cost of additional storage for each user.

Key Share Deletion

Over time, individual key shares in the cloud are independently deleted by the cloud provider. This process can occur at regularly scheduled time intervals, such that a random share is deleted every day, for instance. If user B had locally cached all decrypted key shares, then B will continue to be able to decrypt data from the cloud until his cache needs to be refreshed, or the entire key store expires.

Suppose that key share $KS[1]$ is erased and that B needs to re-fetch key shares from the cloud. B will find that $EKS[1]$ (the encrypted version of KS[1]) is no longer available, and so another key share must be randomly chosen from the available set. B will be required to use an appropriate access key in the set $AK[x]$ to $AK[y]$ to access another available key share outside of the initial three, such as $KS[4]$. Any users that hold the same deleted share may also need to do the same. If three key shares of the initial set of five are deleted, then user B cannot satisfy the threshold, and will be unable to decrypt the user data. If B holds no other access keys, he may optionally obtain $AK[z]$ from A, where z is an access key that was not previously held, and which unlocks a valid remaining key share from the cloud. B may obtain this key from A or from another user in U_{tr}. Otherwise, B must wait for the key store to expire and new valid key shares to become available.

Through the process of gradual share deletion in the cloud, users eventually lose access to the cloud data as a consequence of falling below the share threshold. In this way, revocation is handled without the participation of the data owner. Authorized users may obtain additional shares or wait for share re-generation in order to retain access; discontinuity of access can therefore occur as a side effect for authorized users, but it is temporary.

Keys may be deleted according to different schedules, such as based on regular time intervals, or based on the number of accesses of the user data, or the number of joins and leaves of users in the user set. The size of the valid remaining key store in the cloud will decrease from the initial maximum until the store is re-generated. Access keys must only be re-generated for shares that belonged to users whose access rights were revoked from the last time that shares were generated. In other words, if no user left the authorized user membership U_{tr} in the last round, then key shares will require replacement after deletion, but the access keys held by users need not be updated. To effect control over access key replacement, an expiration flag may be set by the data owner for each access key, if the key unlocks a share that was assigned to a user whose access rights have been revoked; the tradeoff made is between additional record-keeping and the computation of new keys.

Key Share Replacement

Once the number of outstanding valid shares in the cloud subject to random replacement decreases to t, it becomes impossible for all users to download sufficient valid shares to replace those that are deleted, even if additional access keys are obtained from the data owner or other users. The key stored in the cloud then expires; this event can also occur at a prearranged point before the threshold is reached. The content owner A can then proceed to replace the deleted shares in the cloud with newly-generated valid shares of a new version of the symmetric key K'. A new access key will also be generated for each share, or set of shares, to protect them from the cloud provider, unless the corresponding key share did not belong to a user revoked in the last round. Thus, for instance, key $AK'[1]$ will protect the new key shares $KS'[1]$ to $KS'[5]$, from 1 to 5, and so on; a total of n key shares are again stored in the cloud with headers reflecting the new version. In this case, the user data that is stored in the cloud and encrypted with the older key K must be replaced with a version that is encrypted with the new key K'; this may be done by user A, or by a trusted manager M. To avoid a key consistency issue, the ciphertext may be appended with information on the key version required to decrypt it. A new cycle of the interactions shown in Table 6.2 is repeated, for the new version of the data key. In the special case where user data is time-sensitive in nature, such as a stock market transaction, no replacement of key shares may be undertaken. Rather, all users eventually lose access to the data due to share deletion, at which point it is made public or has no intrinsic value.

Revocation

Suppose that user A decides that B should no longer have access to the encrypted user data stored in the cloud. B will be unable to obtain additional access keys from A to obtain more shares, will be unable to obtain updated access keys once the key store expires, and thus will be unable to decrypt shares of the new key K' from the cloud. B will not immediately lose access rights to the originally given shares; he will eventually do so when sufficient shares are deleted non-deterministically by the cloud provider and B's key cache cannot be refreshed. To conserve communication, it is unnecessary for B to poll the cloud provider to discover a new key version to replace an expired one. B may issue a request for new key shares when they become available, and have the cloud provider send them in response in asynchronous push-based fashion, while B is still authorized.

6.3.2 Discussion and Analysis

The main use envisioned with the proposed scheme is the management of continuous access to encrypted user data in a highly scalable manner with minimal coordination, as users join and leave the system. Another use is in allowing access to user data for a finite period of time. In the latter case, once insufficient key shares remain, the encrypted data stored in the cloud is no longer useful to the authorized user set. The data owner will not re-generate a new data key, and the ciphertext c may simply be discarded at the end, as it cannot be decrypted with the remaining shares; another possibility is to simply reveal the data key to the cloud provider and allow global unrestricted access to the user data, as it is no longer considered sensitive once enough time has passed.

The main advantages of the technique are summarized:

1. Expensive key re-generation and re-distribution does not occur every time a single user's access is revoked. The revocation process relies upon the gradual but predictable disappearance of key shares, incurring no cost for the data owner upon each such occurrence.

2. Storage of key shares in the cloud takes advantage of the cloud provider's high availability, which is not the case with techniques relying upon the distribution of shares across a peer-to-peer network. The key shares may be replicated onto multiple cloud providers, if available, to increase redundancy.

3. The key re-regeneration process is not required to be undertaken by the data owner (user A in the example); once sufficient key shares have been depleted, a trusted manager M may be employed to perform it, if such an entity exists within the system, thus removing the communication burden from the data owner.

4. Since the key shares are securely stored in a centralized cloud location, the rate of share deletion is deterministic. It may even be dynamically controlled by the data owner through appropriate instructions to the cloud provider; key shares can also be forced to expire at any time if a security breach occurs. Caching of key shares for a mobile device user is limited by local storage, and thus it is expected that a user will need to fetch shares from the cloud regularly. Regular back up of shares by a user to another server for safekeeping would entail a prohibitive communication cost.

Key shares are stored securely in the cloud. Even if the provider attempts to collude with an authorized user, every key share is protected with a unique access key, and the access keys are never shared with the cloud provider; the user is unable to glean any new and useful information from the cloud provider. If a user shares an access key with the cloud provider in an unauthorized manner, then only a limited number of key shares will be temporarily accessible to the provider until they are deleted; even if the secret data key K is decoded by the provider, it will eventually be rotated and the ciphertext will be replaced. The same safeguard applies to key material stolen from a compromised user. Also, because key shares are encrypted, then even if the provider is malicious, disobeying deletion rules and retaining them in the cloud is uneconomical to the provider and confers no benefit other than prolonging revocation until the next key update.

It is impossible to predict with certainty at what point a user will lose access to key material such that the user will be unable to refresh his or her local copy of key shares from the cloud, as the deletion of any particular share is normally non-deterministic. This is not of great concern, however, as it is likewise impossible to know when a device user will need to update his or her local cache of key shares. From basic probability theory, the chance that any user will have access to a sufficient number of shares is given in Equation 6.1:

$$\left(\prod_{0 \leq i \leq d} \left(\frac{n-i-w}{n-i} \right) \middle| 0 \leq d \leq (n-t) \right) \tag{6.1}$$

where d is the total number of deletions thus far, and w is the amount of shares generated for the user in question, assuming in this case that it is equivalent to the threshold $t+1$.

A compromise in the proposal is the cost of storage on the user's device, not only of key shares, but also of their corresponding access keys; the costs borne are shown in Table 6.3. It is possible to configure the protocol such that an access key is required for each key share, or an access key unlocks more than one share. The advantage of the latter method is a lesser storage penalty; the disadvantage is that a user may continue to have access to a threshold number of shares for longer, i.e. throughout a greater number of share deletion events, adversely affecting the scheme's revocation efficacy. In any case, the key material storage cost is not high, and does not vary with the size of the data being protected.

If all key shares of a data key are stored in a single cloud provider's data store, as in the default scenario described, then the access keys are necessary to prevent the cloud provider from reconstructing the data key itself from a threshold number of shares. Even

Key material	**Size**	**Assumption**
Each access key.	16 bytes	128-bit AES.
Each key share.	16 bytes	128-bit equivalent.
Each encrypted key share.	16 bytes	128-bit AES.

Table 6.3: The cost of storage of key material in the key-sharing model.

if the shares were to be distributed across multiple cloud providers, then less than the required threshold number would need to be stored on each if access keys were not used; in addition, the cloud providers could not collude. As a disadvantage, the data owner bears a significant cost in distributing access keys to recipients; however, this cost may be reduced by storing the access keys in the cloud for dissemination, encrypted with the recipient's public key (if a supporting infrastructure such as RSA is available), so that only a one-time upload of all access keys by the data owner is needed. Another possibility is to employ a trusted intermediary, as in previous chapters, for the purpose of access key generation and distribution. The initial key distribution costs are therefore no worse in this model. Also, the data owner does not need to retain access keys after they are distributed.

A consideration is the communication overhead of recipients requesting multiple key shares, the amount influenced by the threshold value; these requests are made directly against the cloud, however, and the responses containing the shares may be bundled with the downloaded user data to minimize overhead.

Continuous share expiration results in users undergoing access revocation at different times; hence, resulting communication sessions are evenly distributed over time. If all shares expired simultaneously, then users would make requests for additional shares all at once, which could result in a flooding of the network or of the data owner.

An alternative to the proposed scheme is a coarser-grained approach where multiple data keys are stored in whole in the cloud, rather than shares of these keys. Each such key may be assigned to multiple users, and the keys may still self-erode. This modification reduces the complexity of the overall scheme and does not require the expense of reconstruction of the data key from shares by each recipient. However, such a scheme is less flexible for the following reasons, which may make it unsuitable for some systems:

1. It does not permit different portions of key material (i.e. shares) to be stored in different locations such as across multiple cloud providers; such a deployment may be

useful in preventing a cloud provider from reconstructing the data key in case access keys are compromised, because insufficient shares may be stored on its premises and collusion with another cloud provider to obtain more shares is impossible.

2. It does not permit some users to be assigned a greater amount of key material initially, which is useful in supporting prioritization of users, as is suggested in a variant below.

3. Authorization for a user depends on having access to a whole key, and re-gaining access after deletion requires obtaining a new key. In the case of key shares, however, users may incrementally grow their key material through requests for additional shares from the cloud provider, the data owner, or other users; permission for doing so may be controlled dynamically. To achieve information-theoretic secrecy, the key shares must be of length at least equal to the data key itself that is fragmented; however, if computational secrecy is sufficient, then this restriction is relaxed and significant storage cost savings are achieved: the size of the key share store grows by a factor of the number of shares divided by the threshold [65]. Thus, the overall amount of data transferred on key downloads when utilizing key shares may be less.

Even with the use of whole keys, the concept of self-eroding key material under the control of the cloud provider remains a significant contribution of the proposed work.

6.4 Variants

The basic scheme presented may be optionally extended with the following variants if additional network components are present, to improve its performance or reliability:

Priority Classes of Users

Shares need not be evenly distributed across all users. Higher-priority or more trustworthy users could retain key shares for longer; one way to accomplish this is to assign a different class of shares to these users such that the shares undergo a slower rate of deletion than do regular shares assigned to the rest of the user population. Another way is to simply assign a greater number of shares to priority users so that they are less likely to suffer the effect of insufficient shares prior to share re-generation; furthermore, the lock-out of

access due to insufficient shares in this case is shorter-lived. The relative priority of a user class will dictate the rate of deletion of its shares, or the number initially assigned to each represented user, depending on the variant chosen.

Distribution Across a Cloud-of-Clouds

Recently, cloud brokering has been introduced, permitting access to a *cloud-of-clouds*. The advantage is that a client need not rely upon a single cloud infrastructure to provide reliable and continuous service. A broker acts as an intermediary, arbitrating between multiple providers. To take advantage of this concept, key shares may be distributed across the repositories of different providers. Each user is given a range of access keys to shares of a single provider to retain communication efficiency. However, other users will be given access to shares stored in another provider; if one is adversely affected by an outage, it need not disrupt service to all users, thus improving the reliability of the system.

Manager-Assisted Key Shares

The protocol described thus far relies on the data owner re-generating new key shares for storage in the cloud. Although this permits the key management system to rely only on the data owner and no additional network entity, the data owner may be subject to occasional unavailability due to its mobility. A highly-available trusted manager could be a beneficial addition to this scheme. For instance, suppose that key shares must be generated or re-generated. User A, the data owner, would generate the ciphertext c and description key L; the manager would then create the key shares and access keys that unlock them. The manager would upload the key shares to the cloud, and the access keys to all users in U_{tr}. When the shares would expire and need to be re-generated, A would request the manager to repeat the procedure with a new batch of key material; full trust in the manager is required.

6.5 Implementation

6.5.1 Performance Measurement

A prototype of the proposed scheme was implemented to assess its performance. The implementation was realized on popular existing commercial platforms: the Android mobile and the Google App Engine (GAE) cloud platforms. Refer to the system model in Figure 6.3. The Shamir Secret Sharing in Java (SSSJ) library [107] provided the implementation of the underlying cryptographic algorithms; in particular, it implements the LaGrange Interpolating Polynomial Scheme [97]. The implementation was run on different clients to assess their relative performance: an Apple iMac 3.4 GHz quad-core Intel Core i7 desktop computer with 16 GB RAM, an Apple MacBook Pro 2.2 GHz dual-core Intel Core 2 Duo with 4 GB RAM, and a Google Nexus One phone with a 1 GHz Qualcomm Scorpion processor with 512 MB memory. The AES cryptographic algorithm was provided by the Java Cryptography Extension (JCE) library. 128-bit AES keys were used as the access keys used to encrypt and decrypt the shares of a 112-bit data key. Performance benchmark results are shown in Table 6.4, for 100 key shares generated with a threshold number of 5. On the server end, an F1-class front-end instance was run as a Java servlet on the GAE cloud, configured at 600 MHz processing and 128 MB of RAM. A connection was established between the desktop or mobile Android client and an instance running on the GAE cloud via HTTP requests, using JSON for data interchange and the Google Gson library for marshalling between Java objects (used by the Java client and server implementations) and the JSON representation. Note that the security model of GAE does not allow direct network connections and native code execution. Only a subset of the Java 2 Standard Edition (J2SE) SDK 1.6 classes are whitelisted on Android and GAE; fortunately, the required JCE classes are supported. The performance data from the GAE server logs are shown in Table 6.5. The benchmark results show that the processing demands on the mobile device and cloud server are not onerous at all.

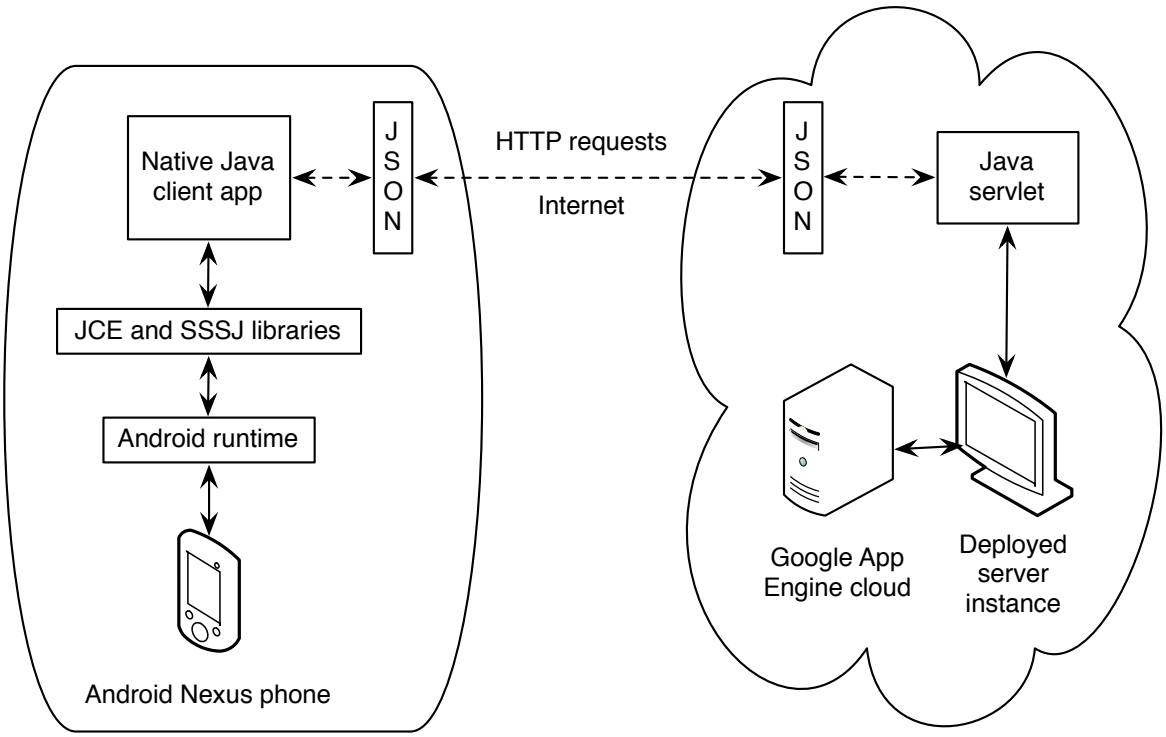


Figure 6.3: A high-level model of the implementation of key-sharing.

6.5.2 Simulation

Through an additional custom simulation program written in Java, various parameters were modified to understand their effect on performance. One effect studied was the rate of share depletion on revocation. Starting with an initial population of 10,000 authorized users, 100 total shares, and a minimum threshold $(t + 1)$ value of 5, shares were randomly allocated to each user. The initial number of shares allocated was increased by a factor of five across the trials. One random share deletion occurred per round of the simulation; users were not allowed to request additional shares and a user became unauthorized once his or her remaining shares fell below the threshold. The results are found in Figure 6.4. Increasing the initial allocation of shares results in a delayed need for additional shares from users at peril of becoming unauthorized, at the cost of delayed revocation and greater up-front storage and communication for key exchange.

Cryptographic operation	Desktop	Notebook	Mobile
Generation of encrypted key shares.	42 ms	56 ms	617 ms
Decryption of encrypted key shares.	$\doteq$ 0 ms	$\doteq$ 0 ms	22 ms
Decryption of encrypted user message.	8 ms	46 ms	163 ms

Table 6.4: The client performance results obtained from the key-sharing implementation.

Cryptographic operation	Response time	CPU time	Response size
Upload of encrypted key shares.	98 ms	38 ms	-
Download of encrypted key shares.	36 ms	38 ms	11.6 kb

Table 6.5: The server performance results obtained from the key-sharing implementation.

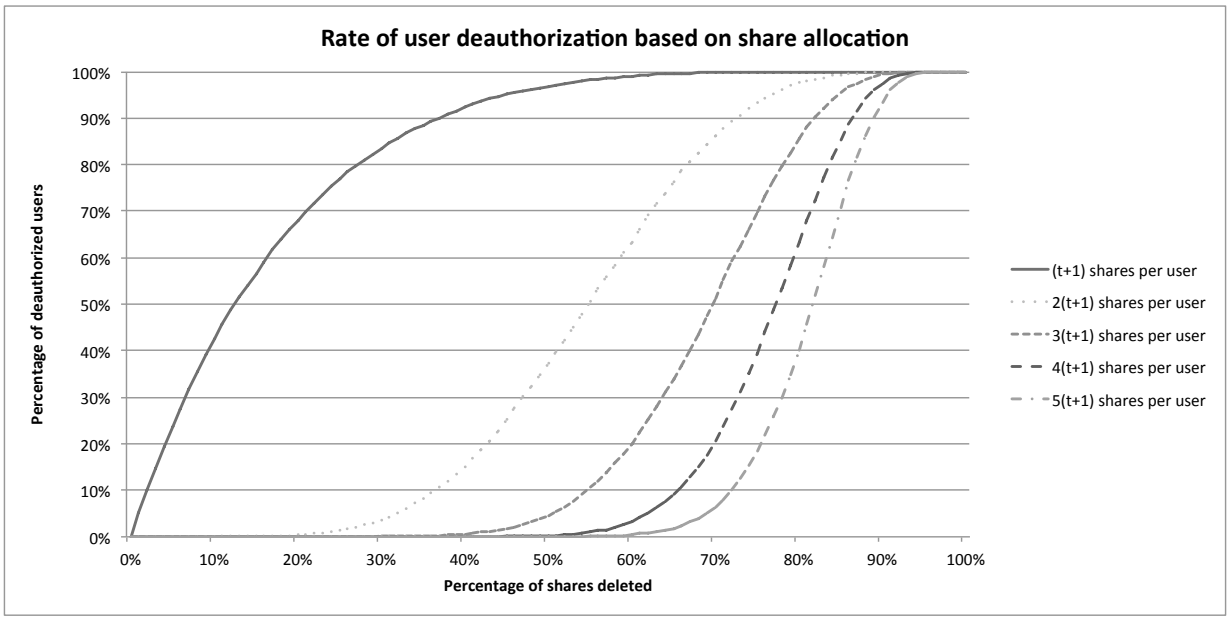


Figure 6.4: The rate of user deauthorization based on share allocation in key-sharing.

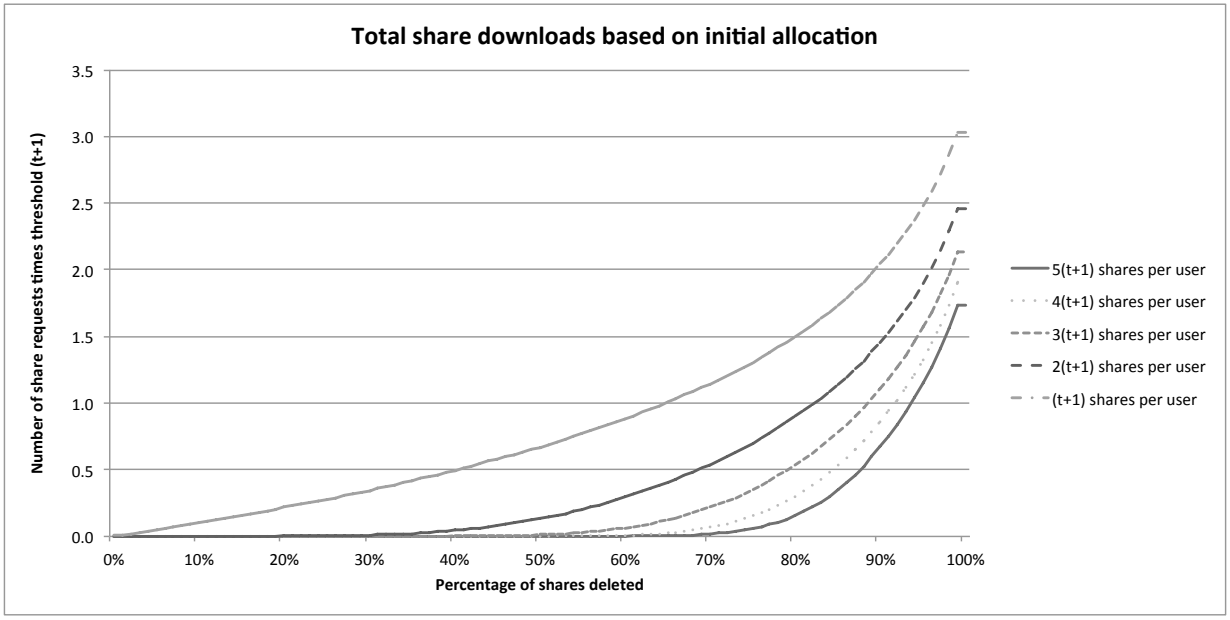


Figure 6.5: The total share downloads based on the initial allocation in key-sharing.

It is also instructive to study the effect of allowing users to request additional shares so that they meet the threshold, assuming that they remain authorized at all times. Keeping the same starting parameters, the initial allocation of shares was varied across the trials from a multiple of five of the threshold amount to a multiple of one; an unlimited number of random shares was allowed to be requested by each user. Regardless of the initial allocation of shares, a request was only made if the user fell below the threshold. The results are shown in Figure 6.5; only the requests subsequent to the initial assignment are shown. There is a trade-off between the amount of shares initially allocated, and the amount of shares requested later to maintain the threshold. Assigning fewer initial shares is more optimal in terms of the average shares requested per user at the end, due to the replacements being randomly drawn from a pool consisting of shares that are still valid.

The rate of key re-generation affects the minimum number of valid users present at any time in the system. Keeping the same starting parameters as in the first experiment, with five shares initially allocated to all users, the number of rounds between key re-generation (and re-population of all shares) was varied; the minimum number of valid remaining users was observed as shares were randomly deleted. The results appear in Figure 6.6. When the

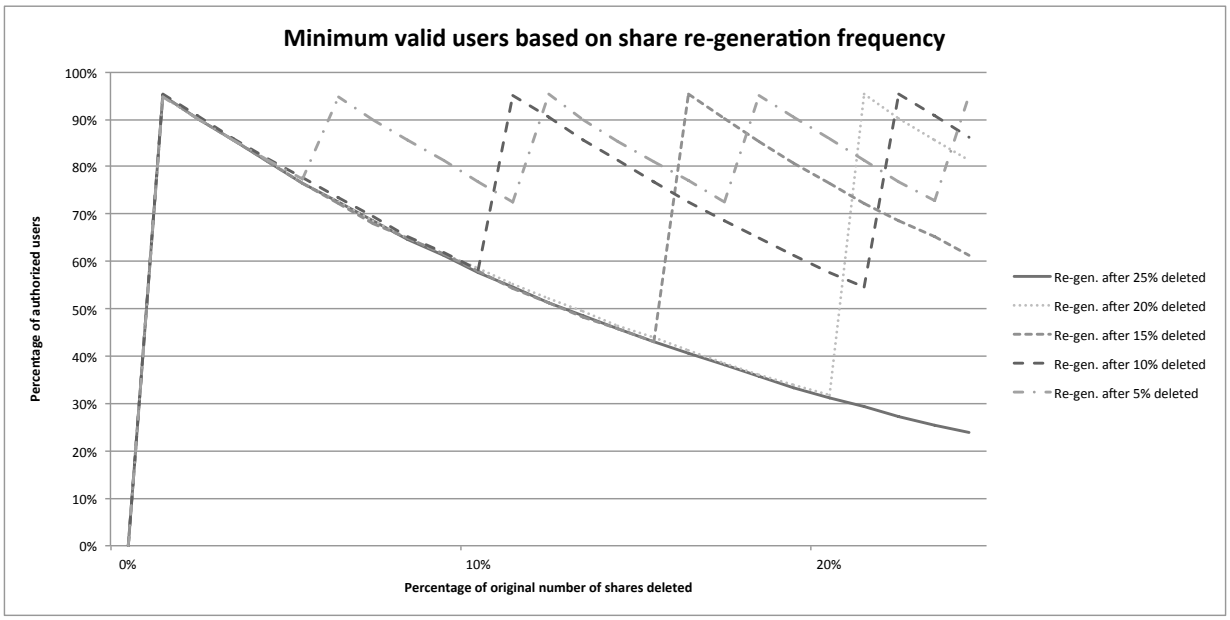


Figure 6.6: The minimum valid users based on the re-generation frequency in key-sharing.

key shares lasted for 5 rounds until they were all re-generated (i.e. after 5% of all shares were deleted), the number of valid users always remained above approximately 72%; when the key shares lasted for 25 rounds, equivalent in number to the threshold share value times five, the minimum number of valid users at times dropped to approximately 21%. Thus, increasing the frequency of key re-generation results in fewer authorized users being locked out, at the cost of more frequent downloads of valid key shares.

Refer now to Figure 6.7. In the case where key shares lasted for 5 rounds, each user requested approximately 18 times the initial threshold amount of 5 shares over a run of 100 deletion events, compared to only approximately 5.1 times the amount of 5 shares where key shares lasted for 25 rounds (i.e. five times longer). Although more frequent key re-generation decreases access wait times for authorized users, it results in more frequent share downloads, which are costly.

If the user population is segmented into multiple classes of different priorities, then the relative rates of user de-authorization will vary. In one experiment, the population was evenly distributed across five priority classes with various initial share allocations; users of the highest priority level 5 were assigned five times the number of shares of users of priority

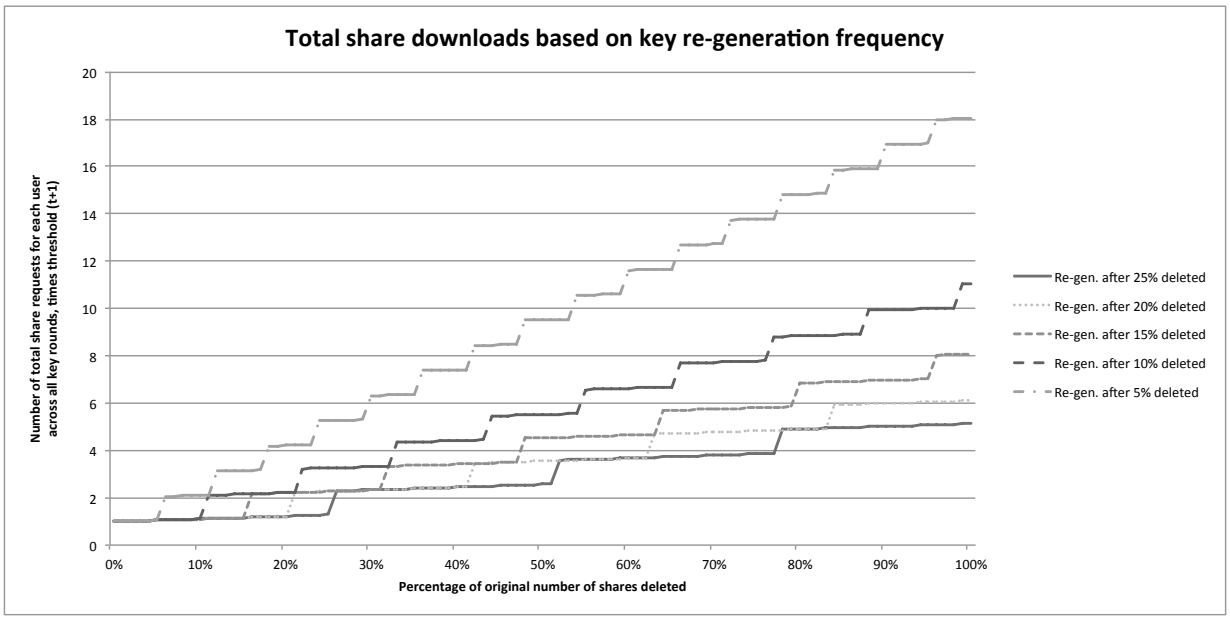


Figure 6.7: The total share downloads based on the re-generation frequency in key-sharing.

level 1, and so on. The key and all shares of it were re-generated every 75 rounds, and additional share requests were disallowed; the effect on the number of outstanding users in the system is observed in Figure 6.8. In the lowest priority class, all users typically lost authorization before the next re-generation round, while in the highest priority class, only approximately 20% lost authorization. Thus, although higher priority classes incur higher initial share downloads, they can retain near-constant authorization to the system and less interruption to their service; because they tend to be of higher trust, it is appropriate for their revocation to be delayed till the next key re-generation. Observe, however, that even if an authorized user, irrespective of priority, loses access to shares initially assigned, then it is permissible for that user to obtain additional access keys from the data owner or a trusted manager to regain a threshold of shares and maintain continuous access to encrypted data in the cloud; such requests were omitted from the experiment in question.

The various parameters discussed in the simulation results should be tweaked as appropriate to fit the user application that is being secured.

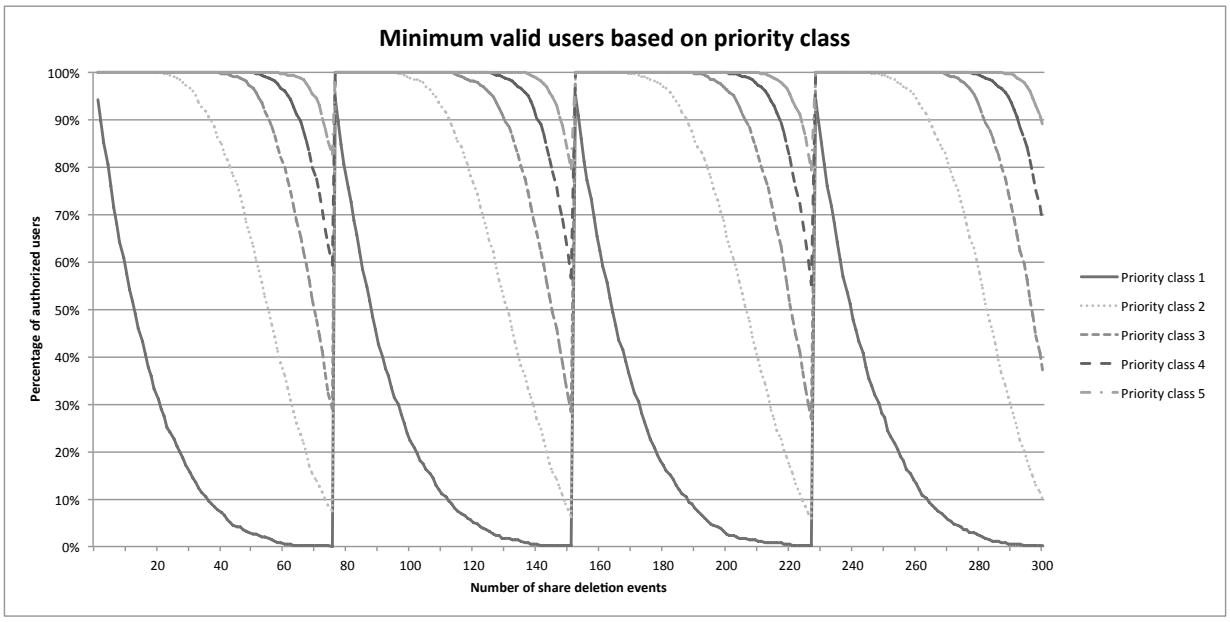


Figure 6.8: The minimum valid users based on the priority class in key-sharing.

6.6 Summary

It has been demonstrated that scalable key management may be attained by leveraging the inexpensive storage capacity and high accessibility offered by a cloud provider. Through the use of self-eroding key shares in the cloud, efficient revocation is achieved for mobile device users without requiring the involvement of a data owner or trusted intermediary. The cloud provider does not have direct access to unencrypted data since it is unable to generate decryption keys itself as the shares are protected through access keys. All key material is stored securely and cheaply in the cloud, rather than among the user population itself. Access by mobile users is accomplished through a direct connection to the cloud and is enabled solely through the possession of appropriate access keys to unlock the shares.

One of the benefits of using centralized and reliable cloud storage for key shares is that there is full control over share management; it is not subject to outside factors such as user churn. Various heuristics may be implemented to control the timing and rate of share deletions, to fit the mobile application being executed. Options exist for different cloud usage scenarios including the case of different priority classes of users being present.

Chapter 7

Query Privacy for Location-Based Services

7.1 Introduction

Prior chapters were concerned with key management methods that guarantee that data outsourced to a cloud server is always kept confidential from the cloud provider. This chapter considers the case where information stored in the cloud is public or permitted to be shared with the provider. This scenario has numerous practical uses: the cloud provider may continuously update its database from external sources, and perform various work on it such as sorting, filtering, and transforming it as required by the application, taking advantage of the cloud's scalable means of computation. However, confidentiality remains a concern for users. The specific information retrieved from the cloud can sometimes reveal much about a user to the cloud provider, beyond his or her identity, and to his or her risk and detriment. Hence, a scheme that allows mobile users to retrieve information in a confidential manner is presented in this chapter such that query privacy is assured. The chosen application context is a location-based service, which is a highly popular cloud application that enjoys significant network effects, where participating mobile users can increase the value of the service to others, and effectively illustrates how the proposed techniques may be applied in practice; other application contexts are possible, however.

The privacy requirements for location-based services are detailed in Section 7.2. Related work on query privacy is described in Section 7.3. The proposed solution for private

information retrieval is detailed in Section 7.4. An implementation on multiple platforms is discussed in Section 7.5, and the work is summarized in Section 7.6.

The content of this chapter is based on work that has been published [88, 89].

7.2 Privacy Requirements of Location-Based Services

Users of mobile devices frequently have a need to find POIs (Points Of Interest), such as restaurants, hotels, or gas stations, in close proximity to their current locations. Collections of these POIs are typically stored in databases administered by LBS (Location Based Service) providers, and are accessed by the company's own mobile client applications or are licensed to third-party independent software vendors. A user first establishes his or her current position on a smartphone through a positioning technology such as GPS (Global Positioning System) or cell tower triangulation, and uses it as the origin for the search.

The problem is that if the user's actual location is provided as the origin to the LBS server, running in a cloud, which performs the lookup of the POIs, then the server will learn of that location. In addition, a history of locations visited may be recorded and could potentially be used to target the user with unexpected content such as local advertisements, or worse, used to track him or her. The user's identity may be divulged through the inclusion of the originating dynamic IP address, e-mail address, or phone number in requests to the LBS server so that the results of an LBS query can be routed back to the correct user via a TCP data connection, e-mail reply, or SMS reply, respectively. If a location can always be correlated to each request, then the user's current pattern of activity and even personal safety is being entrusted to a third-party, potentially of unknown origin and intent. Although search engines routinely cache portions of previous queries in order to deliver more relevant results in the future, concern occurs when the user's exact location history is tracked, and not just the key words used in the search. For many users, this constitutes an unacceptable violation of privacy, and efforts should be made to avoid it.

As location technology becomes commonplace, users will become increasingly aware of and become concerned about location privacy. Not only are privacy and personal safety important considerations, but recent advances in mobile advertising have even opened up the possibility of location-based spam. The challenge has been to design a system whereby a user can retrieve useful POI information without having to disclose his or her exact

location to a third-party such as the LBS server running in the cloud. The user should also not have to reveal what particular POIs were searched for and found, as each POI record typically includes precise location coordinates. Thus, the server should be unable to infer the user's current location or likely destination, or accumulate a history of requests made for profiling purposes. Generally speaking, a user will typically be comfortable with a certain degree of privacy, meaning that the user could be expected to be anywhere within a certain geographic area, such as a city or neighbourhood, without fear of discovery.

Today's smartphones have processors that are suitable for cryptographic operations that can enable location privacy. However, these devices have limited memory and bandwidth. For instance, typical 3G smartphone limits are 128 MB of dynamic RAM, 32 GB of flash memory, and operation on 3G wireless networks no faster than the (theoretical) 7.2 Mbps HSDPA network. Consider these data limits with respect to a typical commercial POI database for the U.S. and Canada, which can contain 6 to 12 million entries and require 1 to 2 GB or more of flash data storage. Requiring that the smartphone download the entire database for each request so as not to provide information about its current location is clearly not practical [102]; nor is requiring that it periodically download just the updated data to ensure accuracy of results, given practical bandwidth limits, data usage limits, and associated overage charges (penalties for exceeding the limits) of smartphone data plans. Thus, it is desirable to provide a cryptographic way for a mobile user to request local information while preserving location privacy. Although extra server-side processing demands must be anticipated on a privacy-enhanced LBS server in the cloud, they may easily be scaled in a cloud computing system at a reasonable tradeoff.

The system model described in Section 2.2.2 on page 14 and the adversary and threat model described in Section 2.4.1 on page 19 are assumed, with the exception that a trusted intermediary is not present, and the cloud provider retains full access to data, which the user does not consider confidential itself. In addition, the cloud provider may collect sensitive personal information about a user, including the user's identity and content of requests, unbeknownst to the user; this may constitute a violation of privacy and is considered a threat against the user, depending on the amount and accuracy of information gathered. The LBS server and cloud provider are still considered generally honest-but-curious.

7.2.1 Requirements and Assumptions

The basic scenario entails a mobile device user operating a smartphone with location technology and wireless data transfer capability. The user searches for nearby POIs (i.e. nearest neighbours) by first constructing and sending a query to a known LBS server running as a cloud application, over the wireless network. The server retrieves the query, performs a search of its POI database, and returns a set of results to the user containing all POIs found in the specified region. The proposed protocol should meet the following requirements:

1. The LBS server must not learn the user's exact location. It may only identify a general region that is large enough, in terms of area and the number of POIs it contains, to confer a sufficient level of privacy to the user's satisfaction.

2. There must be no third parties, trusted or otherwise, in the protocol between the user and the server.

3. The implementation must be computationally efficient on hardware, such as a smartphone, which is resource-constrained. A user may be expected to tolerate a delay of no more than several seconds for any kind of query.

4. The approach cannot rely on a secure processor that is not typically found on a commercial smartphone.

Clearly, these requirements present the need for a mechanism to directly retrieve information in a secure and private way without revealing the contents of the query results, and without the need for an intermediary between the user and the database server to provide some kind of a masking function. Fortunately, there is a branch of cryptography that is associated with retrieving information from a database without revealing which item is being retrieved; it is known as Private Information Retrieval (PIR) [25]. The proposed solution in this chapter is sufficiently generic to allow an application to rely on any PIR scheme. The same assumptions are made as those of the underlying PIR scheme where retrieval is either by object index or keyword [27]. A server is described that can find the relevant POI entries based on the user's location of interest included in the request; this is possible because the entries in the POI database are indexed by their location.

Although PIR satisfies the baseline privacy constraints described, current implementations of it fail to satisfy the third condition, which is usable performance on modern

smartphone hardware. The challenge has been to complement PIR with a new algorithmic approach that effectively reduces the amount of computation without significantly sacrificing the user's location privacy.

Note that no effort is made to hide the user's identity from the location-based service. It is assumed that it is acceptable to reveal the user's identity for the purpose of routing the response to a location-based request, and for offering a customized LBS experience. A user that also wishes to hide his or her identity to some extent may wish to make use of an onion router, such as Tor [36]. However, it is noted that there are application domains where the protection of a user's location using the proposed technique is superior to anonymizing the user's identity. For example, it is easy to try to identify a user who made a query with a particular geographical coordinate, simply by looking up the user who lives at the corresponding residential address and assuming the request did not originate elsewhere. On the other hand, the proposed technique hides query contents from the LBS, and leaves no useful clues for determining the user's current location.

When a typical mobile phone accesses a third-party LBS provider through a wireless 3G or 4G data connection, it is assumed that it reveals only its identity and the query itself to the provider. Unavoidably, a mobile communications carrier is always aware of the user's location based on the cell towers in contact, and so it must not collude with the LBS provider; this assumption relies on the LBS provider not being integrated into the carrier's infrastructure, such as a traffic reporting service using cell tower data that discovers a user's location passively. This assumption is valid for the vast majority of LBS applications, which are unaffiliated with the carrier; these include search portals, social applications, and travel guides. When communicating with such an application, the mobile user's IP address is of no help in determining the user's physical location, as it is dynamically assigned independent of location. Only a central gateway that is administered by the telecommunications carrier will be identified. It is assumed that no other information will be gleaned by the LBS provider. In the case where a mobile user utilizes Wi-Fi instead, the user will be assigned an address that points to the nearby access point, however, and may need to employ other techniques, such as Tor, to mask the address.

7.3 Related Work on Location Privacy and PIR

A brief overview of cloaking- and PIR-based approaches for location privacy is provided. A survey and classification of methods for location privacy in LBS can be found in [44, 104].

7.3.1 Location Cloaking Techniques

Location cloaking seeks to prevent an attacker from being able to match queries to particular users and to thus compromise their privacy. The attacker may be able to observe traffic flowing through the network or even be situated at the LBS provider endpoint.

One popular cloaking technique is based on the principle of k-anonymity, where a user is hidden among *k-1* other users. Queries from multiple users are typically aggregated at an anonymity server which forms an intermediary between the user and the LBS provider. This central anonymity server can provide spatial and temporal cloaking functions, so that an attacker will encounter difficulty matching multiple queries that are observed with users at particular locations and at particular points in time. Many cloaking solutions for location privacy suggest either a central anonymity server as described [51, 117], or other means such as decentralized trusted peers [29] or distributed k-anonymity [124].

The chief problem is that the anonymity server must normally be part of the trusted computing environment and represents a single point of vulnerability. If it is successfully attacked, or collusion with the LBS server occurs, then the locations of all users may be divulged. It is also observed that although a cloaking technique by itself is advantageous in that it does not result in increased computational cost on the server, it can carry with it a high communication cost from the LBS provider to the client. This can mean a large and unacceptable penalty for mobile phone users. Finally, if a reduced sample population results from the number of active users in a particular geographic area, it may not suffice to satisfy the desired degree of anonymity. If the anonymity server delays execution of a request until the k-anonymity condition is satisfied, then this delay may prove to be unacceptable to the user from a feature interaction point of view.

7.3.2 PIR-Based Techniques

A PIR technique can be used to ensure that queries and their results are kept private. Specifically, PIR provides a user with a way to retrieve an *item* from a *database*, without the database (or the database administrator) learning any information about which particular item was retrieved. PIR satisfies the described requirements for privacy and low communication cost. However, existing PIR techniques have drawbacks of high computational cost for applications that require low latency. The PIR database is typically organized as an n-bit string, broken up into r blocks, each n/r bits long. The user's private input or query is typically an index $i \in \{1, ..., r\}$ representing the i^{th} block of bits. A trivial solution for PIR is for the database to send all r blocks to the user and have the user select the desired block at index i, but this carries a maximum cost of communication and is unsuitable in a resource-constrained environment such as a wireless network.

When the PIR problem was first introduced [25], it was proven that a single-database solution with information-theoretic privacy and a sub-linear communication complexity (between the user and the database) is impossible to achieve. Information-theoretic privacy assures user privacy even for an adversary with unlimited computational capability. Using at least two replicated databases, and some form of restrictions on how the databases can communicate, PIR schemes with information theoretic privacy are possible [15,48]. The first single-database PIR scheme [26] only assures privacy against an adversary with limited computational capability (i.e. polynomially-bounded attackers). The type of privacy protection known as computational privacy, where computational capability is expected to be limited, is a weaker notion of privacy compared to information-theoretic privacy. Nonetheless, computational PIR (CPIR) [26, 67] offers the benefit of fielding a single database. Basic PIR schemes place no restriction on information leaked about other items in the database that are not of interest to the user; however, an extension of PIR, known as *Symmetric* PIR (SPIR) [83], adds that restriction. The restriction is important in situations where the database privacy is equally of concern. The only work in an LBS context that attempts to address both user and database privacy is [45]. Although, not strictly an SPIR scheme, it adopts a cryptographic technique to determine if a location is enclosed inside a rectangular cloaking region; the goal was to reduce the amount of POIs returned to the user by a query. Unlike the approach presented in this chapter, it fails to guarantee a constant query result size which defeats correlation attacks, and it requires dynamic partitioning of the search space which may be computationally intensive. It also requires two queries to

be executed, whereas a single query-response pair is sufficient in the proposal herein.

PIR has been applied to solving the problem of keeping a user's location private when retrieving location-based content from a server database. This content typically consists of POIs, with each entry consisting of a description of a place of interest as well as its geographical location. A related work exists that does not utilize a third party [46], but it differs from the PIR approach in this chapter in three important ways: first, the approach is based on a dated computational PIR scheme [67] that is less efficient than more recent schemes; second, it has a linear computational cost for a large number of entries, which is too costly for low-bandwidth devices, without allowing users to specify a desired level of privacy; third, it does not consider a privacy-preserving partitioning approach for the data set. In contrast, the work in this chapter will be shown to use partitioning of POI data to permit cloaking, and offers privacy protection when used in conjunction with PIR.

Most of the PIR-based approaches for location privacy rely on hardware-based techniques, which typically utilize a secure coprocessor (SC) at the LBS server host [4, 53] to realize query privacy; a major drawback is that it requires the acquisition of specialized tamperproof hardware and it usually requires periodic reshuffling of the POIs in the database, which is a computationally-expensive operation [4, 56].

7.3.3 Hybrid Techniques

Hybrid techniques [44] permit privacy-efficiency tradeoff decisions to be made by combining the benefits of cloaking and PIR-based techniques. A tradeoff between privacy and computational overhead has been conjectured [28] as a means of reducing the high computational overhead for some application areas of PIR. This chapter validates this conjecture in the context of LBS, and shows how to reduce the performance overhead of current PIR techniques. In particular, this chapter answers the open question of how to reduce the processing cost of PIR without requiring the use of multiple CPUs to take advantage of parallelization, as discrete parallel processors are not typically found on smartphones.

7.4 Proposed Solution

A hybrid solution was developed that utilizes PIR to achieve query privacy in the context of a location-based service, and a cloaking technique to reduce the computational cost of PIR to a feasible level. The proposed technique essentially describes how the user creates a cloaking region around his or her true location, and performs a PIR query on the contents of the cloaking region only. The benefits are numerous: the user's location is kept hidden from the cloud server to an acceptable degree regardless of the number of other users in the area; there is no intermediary server that is responsible for cloaking and that would need to be trusted; and the computational cost of the cryptographic algorithms employed is still practical. It is ensured that the user downloads only the POIs that are of interest to the smartphone, keeping wireless traffic to a minimum to reduce costs and conserve the battery. The proposed solution is described in this section.

The approach that is proposed entails two phases. First, there is a pre-processing phase in which the system is set up for use. The pre-processing operation must be carried out whenever significant changes are made to the POI database on the server. In practice, it can occur every few months during a period of low usage on the server such as nighttime maintenance activities. Second, there is an execution phase, in which the LBS server responds to queries for POIs. The pre-processing phase consists of the following steps:

1. A geographic region is projected onto a two-dimensional plane.
2. A suitable grid is formed on the plane.
3. A POI collection is saved in a database in the cloud such that each row corresponds to one POI.
4. Each cell of the grid is mapped to a portion of the database, i.e., a particular set of database rows (each containing a POI).
5. The grid structure is transmitted and saved on the client device in a local mapping database so that it can be referenced in a subsequent query.

The execution phase, in which a query is made for a set of nearby POIs, consists of the following steps:

1. The user determines the area of interest, either based on the current physical position as determined through GPS, or some other arbitrary area of interest that the user may be traveling to in the future.

2. The user chooses a desirable level of privacy.

3. The client creates a cloaking region corresponding to this level of privacy, which will enclose the area of interest.

4. The client sends the cloaking region to the cloud server and identifies which portion of the cloaking region contains the area of interest, in a way that is hidden from the server itself.

5. The server receives the request, and finds the database portion corresponding to the cloaking region. A block of rows is retrieved from this portion based on the user's specified location of interest. The POIs present in these rows are transmitted back to the mobile client.

6. The client decodes the result, and automatically finds the nearest neighbour POI, or presents the full list of POIs returned to the user to choose amongst.

7.4.1 Level of Privacy

To defeat a server's ability to narrow down the search space for the item of interest to the user, PIR protocols typically process every item, or POI, in the PIR database. This results in a computational complexity that is linear in n (where n is the number of items in the PIR database). This is the main hindrance to practical PIR deployment [102].

A tradeoff is proposed to make the PIR-based solution practical: users are given the choice of trading off privacy for better query performance, by specifying the levels of privacy that they want for their queries. A level of privacy for the query indirectly determines the number of items that the PIR server must process in order to provide a response. Setting levels of privacy is a common practice in several domains where privacy is important, such as web browsing. In the specific case of location privacy, it is argued that resource-constrained device users are willing to trade off privacy to obtain reasonable performance, i.e. to trade off some levels of performance to gain some levels of privacy support.

A user sets the desired privacy level by specifying a subset of the entire database to be queried. The privacy level can be specified in terms of cities or towns (i.e. city level), states or provinces (i.e. provincial level), and so on, to enhance user-friendliness. The ratio of the number of POIs inside this subregion to the number of POIs in the entire POI database defines a privacy parameter ρ; a value of 1 indicates that the user desires query privacy at the same level as that offered by a typical PIR protocol, which is maximum privacy at maximum performance cost. Similarly, if a user sets the query privacy level to 0.25, for example, then the PIR query will execute faster as the server only needs to process one quarter of the entire database. Although the cost is still linear in the number of items in terms of computational complexity, the constant term is modified (i.e. in terms of Big-O notation), leading to significant performance gains. In this case, the server can only infer that some portion of one-quarter of the entire database is of interest to the user. At the same time, it will be disclosed to the server that three quarters of all items in this case are not of interest; depending on the geographical coverage of the database, this leakage of information will not necessarily constitute a significant breach of location privacy.

The cloaking region is thus identified as a subset of the entire world described by the database. If it is imagined that the world is mapped as a grid of so-called geographic grid cells that are equally distributed, then one of these cells will be chosen to comprise the cloaking region. If a higher privacy level is desired, then the cloaking region may be expanded to include multiple geographic grid cells, and thus a larger portion of the database that describes the world. It is sufficient to identify each grid cell by its cell number if the mapping is static and published. The process of mapping the world to a geographic grid occurs during the pre-processing phase, described next.

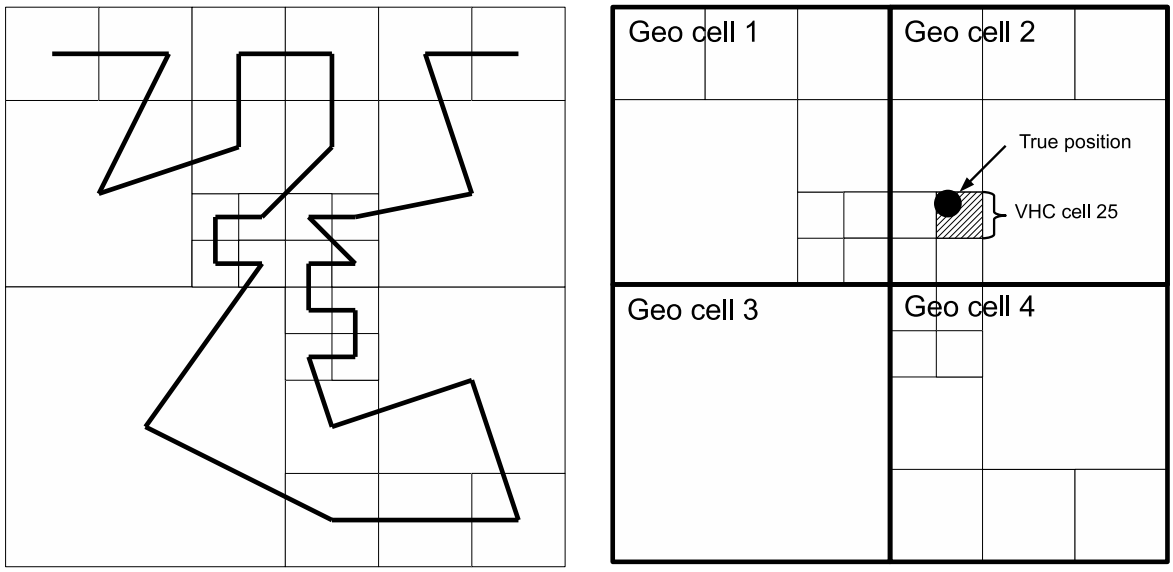
7.4.2 Pre-Processing and Location Cloaking

The first step in the pre-processing phase is to represent a geographic area such as the United States and Canada on a two-dimensional plane using a map projection method such as the commonly-used Miller cylindrical projection [103]. Once that is done, the user's location of interest may be found on this plane. It is necessary to obscure the user's location by creating a cloaking area around the user's true position or area of interest. POIs will be found anywhere by the LBS server within this cloaking region. The cloaking region must be sufficiently large in order to achieve sufficient privacy for the user, but at

the same time it must be sufficiently small to minimize the amount of computation required on the cloud server to process the query results, and to achieve quicker response from it.

Several techniques allow POIs to be mapped to a cloaking region. One technique is quad-tree mapping [51], but it has the disadvantage (from its use in Casper [84]) of forming an unnecessarily large cloaking region which can impair performance [12]. Another technique is called VHC (Various-size-grid Hilbert Curve) mapping [91], which suits the purpose here. In particular, it solves the problem of the density of POIs varying by geographic area. If the density of POIs is significantly higher for a given region (such as a city), then a higher data traffic cost will result if the geographic size of the enclosing cell that is retrieved is always constant, and the query will be slower, as a result. If on the other hand, the density becomes significantly lower (such as in a sparsely populated region like the countryside), then the result size for the queried region may be so minimal that the server may guess the user's likely destination with a high degree of confidence, leading to loss of privacy. VHC solves this problem by creating variable-sized cells from which a cloaking region is built; the boundaries of each cell are established according to the density of the POIs in the geographic area that it encloses. Essentially, in VHC, the two-dimensional geographic grid is mapped to a one-dimensional space such that there is equal POI density in each VHC cell, or subspace, as shown in Figure 7.1 (a). Assume that a typical POI database that covers the regions of Canada and the U.S. will have 6 million POIs. If each VHC cell must contain the same number of POIs, such as 60, then there will be a total of 100,000 VHC cells that will cover this geographic region. Suppose that the lowest POI density found in the database is 60 POIs per 40,000 km^2. Thus, the maximum size of a VHC cell will be 40,000 km^2. Now, a geographic grid is created overlaying the U.S. and Canada regions with fixed-size square cells that are 200 km in length (the area of each is 40,000 km^2). Each such geographic grid cell corresponds to the maximum possible size of a single VHC cell as described above. Each geographic grid cell, however, may contain any number of smaller-sized VHC cells if the POI density of the region inside the grid cell is greater, as shown in Figure 7.1 (b). The decomposition continues until all POIs present within a geographic grid cell are mapped to VHC cells of various sizes and boundaries; each VHC cell will contain the same number of POIs, with empty results padded if necessary.

During normal operation, the client determines a cloaking region based on a particular privacy level by determining the number of geographic grid cells to include inside the cloaking region queried by the user. Suppose that the client chooses a privacy level such that the cloaking region consists of four geographic grid cells. The user's true location is



(a) (b)

Figure 7.1: (a) An example of VHC mapping with uniform POI density. (b) A user's true position inside VHC cell 25 (shaded) and within a cloaking region bounded by the single geo (geographical) grid cell 2. Only the POI results for VHC cell 25 will be returned in a query. If a larger cloaking region consisting of geographic grid cells 1 to 4 was specified, for greater privacy, then the same POI results would still be returned, but at the cost of greater server-side computation and a longer transaction.

in one of these grid cells. Inside of the geographic grid cell, there is a set of variable-sized VHC cells according to the distribution of the POIs in the geographic grid cell. The user's area of interest, for which POIs will be retrieved, will be the single current VHC cell found inside one of the geographic grid cells. The number of POIs per VHC cell is known, and in this case, it is 60. Thus, the user will initiate a request that will reference the cloaking region, as well as the specific VHC cell in which the user is located or interested in. The user will receive a set of 60 POIs that are found in his or her current VHC cell only. The server will only know that the location of interest is somewhere within the cloaking region defined by the four geographic grid cells.

The geographic grid is useful in specifying the size of the cloaking region and for identifying which VHC cells will comprise the cloaking region. The desired level of privacy

Geo cell 1	VHC cell 1	POI 1	POI 2	...	POI 60
	VHC cell 2	POI 61	POI 62	...	POI 120
	VHC cell 3	POI 121	POI 122	...	POI 180
	VHC cell 4	POI 181	POI 182	...	POI 240
Geo cell 2	VHC cell 5	POI 241	POI 242	...	POI 300
	VHC cell 6	POI 301	POI 302	...	POI 360
	VHC cell 7	POI 361	POI 362	...	POI 420
	VHC cell 8	POI 421	POI 422	...	POI 480
	⋮	⋮	⋮	⋮	⋮

Figure 7.2: An example of POI database mapping in PIR, showing the relationship between geographical grid cells, VHC cells, and POIs as stored in database rows in the cloud.

establishes the size of the cloaking region. The client then sends this cloaking region to the server, by identifying the bounding coordinates (i.e., the longitude and latitude of the top-left and bottom-right corners). The server will then be able to identify which VHC cells belong to this cloaking region, and therefore which portion of the database must be read. The client must also encode the identifier of the VHC cell containing the area of interest inside the PIR query. (Each VHC cell in the system is uniquely identified by a numeric value.) Figure 7.2 further illustrates the relationships among a geographical grid, VHC cells and POIs.

Thus, the proposed cloaking technique provides a way of reducing the search space of the POI database by employing multiple levels of database segmentation. The cloaking region itself is described as a single, or multiple, geographic grid cell or cells. Inside each geographic grid cell are found one or multiple VHC cells, the number depending on the amount and distribution of POIs inside the geographic grid cell. The user's true location is inside one of these VHC cells, and the user retrieves POI's corresponding to that VHC cell only. As far as the LBS server is concerned, the user could be located anywhere within the larger geographic grid cell, or multiple grid cells, that comprise the cloaking region.

The geographic grid is fixed. The initial grid cell dimensions are configured based on the maximum size of each VHC cell, but once established, will not need to change. Both the client and server must have the same knowledge of the geographic grid. It can be distributed offline, along with the software for the user's smartphone, or stored in a public directory in the cloud. A simple approach to determining grid cell dimensions is to use a geographic coordinate system such as Degrees-Minutes-Seconds (DMS) [62]. For instance, each grid cell may be two latitude degrees in length, which roughly equates to 200 km at the 30 degree latitude. A population of tens of thousands to millions of users may typically inhabit and stay within the bounds of a grid cell that is 200 km^2 in size, leading to excellent privacy. Cells of larger size will afford province- and state-level privacy if desired.

Both the client and server must agree on the same VHC mapping, and this mapping must be done off-line in advance. Because it is dependent on population density, it will remain relatively static over time even as the population grows, and can be dynamically updated on the client if necessary. In order to contain knowledge of the mapping to define the cloaking region, the user may make use of a pre-computed map file that is stored locally on the device. This mapping technique is an improvement over a cloaking region that is simply based on cells of constant size, and ensures that a constant and predictable number of results are returned for the user's grid cell, so as not to leak information to the cloud.

The idea of using VHC to address the general problem of location privacy was proposed in [91], but in a way that is very different from that of this chapter. Specifically, VHC was used to map the user's current location to a 1-dimensional space. Random perturbation was then applied on the 1-dimensional value, which was then mapped back to 2-dimensional space according to the VHC mapping, to represent the user's true location. In essence, the random perturbation was applied to create confusion for an attacker about the user's true location. The technique proposed here differs in that VHC is used for a different purpose; it defines the storage of POI entries of interest within a geographic cell, which comprises the cloaking region, in a way that allows proximate POIs to be stored as adjacent database entries. This cloaking region is then utilized within the context of a privacy-preserving PIR protocol. Perturbation of the location is not performed; it is argued that this would result in decreased privacy. Indeed, a non-stationary user whose true location is randomly perturbed is still subject to correlation attack. In the proposed approach, it is demonstrated that the cost of computational and communication overhead through the use of PIR is acceptable, as a method is provided for retrieving only a subset of entries of the entire POI database for each query. The proposed technique is also impervious to correlation attacks.

The device must store a copy of the VHC map in local non-volatile memory, but the storage requirements are very reasonable. The current geographic grid cell encapsulating the user can be derived from the user's current latitude and longitude coordinates, if the mapping convention is known. A single coordinate for the intersection point of each VHC cell inside (i.e. one of its corners) can then be recorded. Hence, a single coordinate would suffice to store each VHC cell in device memory. For quick lookup and to minimize storage requirements, the coordinates of all VHC cells only in the current geographic cell could be stored. Assuming that the smallest VHC cell size is 1 km^2 in size, then the worst case is that 40,000 coordinates will need to be stored to account for all VHC's. Two bytes will be sufficient to store each VHC coordinate, because the origin of the geographic grid cell is known, so that the total cost will be approximately 80,000 bytes to store all VHC cells. This is the worst theoretical case; in practice, small VHC cells will only be encountered in very dense metropolitan areas, and they will not occupy an entire geographic cell.

7.4.3 Variable Level of Privacy

The size of the cloaking region chosen and the subsequent performance of a query depend on the user's desired level of privacy. If the user wishes to obtain a higher level of privacy, then the size of the cloaking region can be defined to be larger, and to encompass a larger number of geographic grid cells (and thus VHC cells found inside), but the amount of computation on the server will increase accordingly, delaying the response. Nevertheless, the chief benefit is that the processing time of the query on the server is predictable, because each VHC cell in each request contains the same number of POIs. The key fact is that the amount of data transmitted will be roughly proportional to the number of POIs in a single VHC cell (depending on the implementation details of the PIR scheme being employed), but the server will only learn the client's location to the resolution of the cloaking region. The amount of variation allowed in the size of the cloaking region should be kept to a minimum, as this variable may be used to form part of a fingerprint of a target in a correlation attack. Allowing a one-cell or two-by-two-cell region only may be a good compromise. The latter could be employed by the user on a permanent basis to avoid the threat of inter-cell movement being discovered.

7.4.4 Algorithm

In this section, the algorithms that implement the proposal to allow a user to set his or her level of query privacy (equivalent to the size of the cloaking region) are presented:

Let $PIR = \{PIREncode, PIRProcess, PIRDecode\}$ be some PIR protocol where $PIREncode$, $PIRProcess$, and $PIRDecode$ are the query encoding, response processing, and response decoding protocols, respectively. The following generic algorithms implement the proposal for introducing levels of privacy in the PIR query.

Query Generation (By PIR Client)

Let n be the total number of items (or POIs) in the PIR database or databases (in the case of a PIR protocol with replicated databases), σ be the number of VHC grid cells in the map where each grid cell has n/σ items. Let i be the index of the database block that the user wishes to retrieve, and $\rho \in [0, 1]$ be a privacy parameter preset by the user that determines the size of the cloaking region.

i. Compute $l = \lceil \rho n \rceil$ and set $R = \{r_1, r_2, r_3, ..., r_l\}$ to be the set of indexes for the items corresponding to the single or multiple geographical grid cells of the cloaking region. For a standard PIR query, it will be the indexes of items in the database.

ii. Compute $q = PIREncode_R(i)$ as the PIR query encoding for i, using only the item indexes in R (i.e. not all the indexes in all geographical grid locations in the entire map are required).

iii. Send $\{q, R\}$ to the database (or PIR server). Instead of sending R, it may be more efficient to send only the top left and bottom right coordinates of the bounding rectangle that covers the cloaking region, or the range of identifiers (or numbers) for the VHC grid cells that are within the cloaking region, or for the geographic cells that contain them; in any of these cases, the PIR server can use the provided information to determine R.

Response Encoding (By PIR Server)

i. Retrieve a database portion $D = \{d_1, d_2, d_3, ..., d_n\}$, where $D[r_j] = d_j \forall j, 0 \leq j \leq l$, from the database. Each item may consist of one (or more POIs) and each POI is a data structure with attributes of longitude, latitude, name, address, phone, category, web site address, and so on.

ii. Execute $PIRProcess_{D'}(q)$ to obtain response r, which is the block of POIs in the user's VHC grid cell, and return it back to the client.

Response Decoding (By PIR Client)

i Execute $PIRDecode_R(i, r)$ to obtain a database response to the query. The response should be the set of POIs that the query requested.

ii The client can locally compute the nearest neighbour using the set of POIs returned.

7.5 Implementation

7.5.1 Performance Measurement and Simulation

C++ and Java prototypes were developed using two available implementations of the PIR protocol. The evaluation of the proposed approach in terms of feasibility and scalability is based on the C++ prototype. The purpose of the Java prototype is to demonstrate the successful porting of the implementation to a smartphone platform. It was not the intention to compare these implementations or run them with the same set of parameters.

The C++ prototype is based on Percy++, an open source PIR protocol written in C++ [47, 48]. The Percy implementation offers computational, information-theoretic and hybrid (a mix of both) PIR. Percy++ was modified to support the proposal for allowing PIR queries to be based on a database portion defined by the cloaking region. The computational performance of the PIR algorithm is measured, taking into account different levels of privacy and corresponding sizes of cloaking regions. The PIR implementation was run against a database of 6 million synthetic POIs, the typical number of POIs in a commercial POI database for the U.S. and Canada. It is noted that a similar experiment

in [46] considers a much smaller database consisting of only 10,000 and 100,000 POIs; a head-to-head comparison with [46] is infeasible because different PIR implementations and test data were used. Each POI consists of 256 bytes that were generated randomly; this size is a conservative representation of practical POI sizes. In comparison, the POIs from [46] are only 64 bits in length. The location coordinates are stored with each POI.

The Java prototype is based on a computational SPIR protocol implementation [96]; this SPIR protocol was derived from an oblivious transfer protocol [85] and appeared to be the only publicly available Java implementation at the time of writing. This second prototype development consists of both a server component and a client component that was deployed on a smartphone platform. Specifically, the implementation from [96] was ported to Google's Android smartphone platform, which supports the Java programming language. The only aspect of the implementation that could not be adapted without light modification was the RMI mechanism, which was replaced with HTTP socket communication between the Android client process and a server process running on a desktop computer.

7.5.2 Discussion

The query roundtrip times were measured for the C++ prototype on a machine with a 2.91 GHz dual-core AMD CPU, 3 GB RAM, and running Ubuntu Linux. Since the Percy++ PIR uses replicated databases, the number of databases was set to 2 [48]. Figure 7.3 shows query roundtrip times for varying sizes of cloaking regions and POI result sizes. The number of POIs returned for each query is equivalent to the number of POIs configured in a VHC cell. Similarly, the number of POIs returned by a query is equivalent to the number of blocks (in bytes) that a traditional PIR query returns. For instance, a block of 10 POIs is equivalent to 2560 bytes of data, as each POI consists of 256 bytes. The query roundtrip or response times for block sizes 5, 10, 25, 50, 100, 250, and 500, where the cloaking region is as large as the entire database itself for maximum privacy, are between approximately 20 and 70 seconds; this is because each PIR request runs against the entire database of 6 million synthetic POIs. However, the query roundtrip time improves with lower levels of privacy. For example, the query response times for the above block sizes with a privacy parameter of $\rho = 0.17$, and thus a smaller cloaking region, are between approximately 4 and 12 seconds. One must observe that setting ρ to 0.17 is equivalent to privately querying a block of POIs from a portion of the database consisting of 1.02

million POIs. If it is assumed that there are equal number of POIs in all the provinces and states of Canada and US, this implies a cloaking region that covers approximately 10 provinces and/or states. Under a similar assumption, a user who intends to hide his query in a cloaking region consisting of one province or state will simply set his query privacy parameter ρ to a much lower value of 0.02. The query response time for this level of privacy is approximately 0.3 seconds for an optimal block size, which in the testing configuration consists of 256 POIs.

It is easy to observe from the graph that the block that consists of 250 POIs gives the best performance. Furthermore, the worst performing block size is the one consisting of 5 POIs, the reason being that smaller block sizes require more rounds of computation to process the individual blocks, compared to larger block sizes. On the other hand, large block sizes, such as 500, carry performance penalties and overheads which depend on the characteristics of the underlying PIR scheme, and also on the resource constraints of the runtime hardware (e.g., RAM, disk and memory cache sizes, and network bandwidth). The network cost was negligible since the measurements were taken on a LAN.

The client for the Java prototype was also installed on a G1 Android smartphone that features a Qualcomm ARM processor running at 528 MHz, and includes 192 MB of DDR SDRAM, and 256 MB of flash memory. Although the locked smartphone was capable of running on T-Mobile's 3G network in the U.S., it did not support the 3G frequency bands in operation in Canada; hence, the tests were run using the Rogers EDGE network, which is slower by up to a factor of ten. An Android application was created with a user interface, shown in Figure 7.4, that allows the user to specify the server address and query parameters such as the size of the cloaking region and the size of the portion of the cloaking region to fetch. It was observed that when the cloaking region was reduced to a quarter of its original size (i.e. a quarter of the POIs were returned), the query generation became 2.15 times slower, but the roundtrip time became 3.32 times quicker. Overall, the implementation was usable even though it had not been originally designed and optimized for the Android platform, and it was restricted to a non-3G network.

The proposed solution preserves the privacy of the user's location irrespective of the number of other users initiating queries for the same location. The server can infer the user's location based on the cloaking region only. The user may adjust the size of the cloaking region based on his or her personal preferences (i.e. the desired level of privacy, query performance, and cost), because a larger region will entail more computation. The

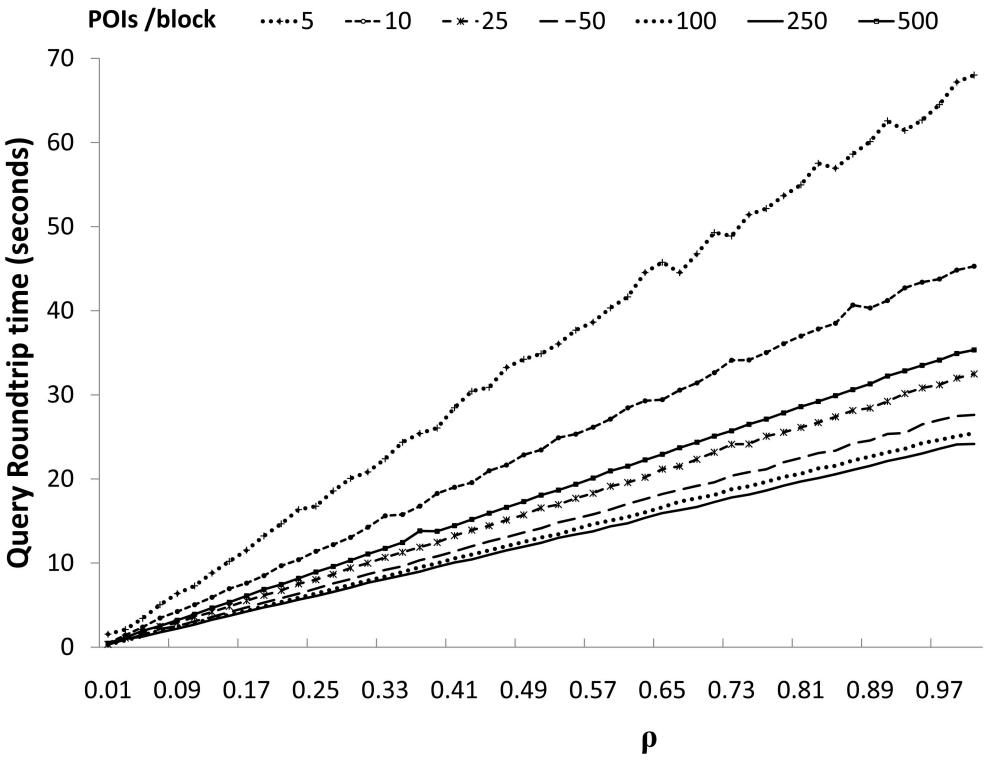


Figure 7.3: The query roundtrip performance results obtained for different sizes of cloaking regions, as determined by the privacy parameter ρ, and for different block sizes of POIs returned per query. A single measurement was taken per data point.

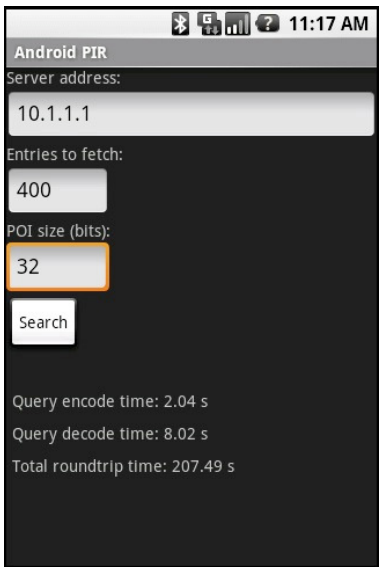


Figure 7.4: The user interface of the mobile client app in PIR.

size of the cloaking region is based on a particular size of geographic area and does not need to be adjusted based on the known distribution of POIs within the region. The user only establishes a reasonable level of privacy based on the number of geographic grid cells that define a geographic area. The boundary of the cloaking region utilized in a request is established by the user and is based on the geographic cell map contained on the user's end as well as the privacy parameter. The size of the cloaking region and its boundaries are dynamically adjustable and are not controlled by the server.

7.6 Summary

In this chapter, an algorithm has been proposed for private information retrieval from a cloud server that achieves a good compromise between user location privacy and computational efficiency. The proposed algorithm has been implemented and evaluated and shown to be practical on resource-constrained hardware. The proposed approach of using a variable-sized cloaking region divided into VHC cells results in greater location privacy than the traditional approach of a single cloaking region, while at the same time decreasing wireless data traffic usage from an amount proportional to the size of the cloaking region to an amount proportional to the size of a smaller VHC cell. It also allows the user to dynamically choose from various levels of privacy. Although increasing the size of the cloaking region does result in higher computation in processing the query on the cloud server, it is maintained that this tradeoff is very reasonable, given that the cloud provider is highly scalable; furthermore, only relevant results are transmitted over a relatively slow and expensive wireless network, and the processing overhead for the mobile user is negligible.

Chapter 8

Conclusions

The works in this thesis represent progress towards securing mobile cloud computing systems in a highly scalable manner. Confidentiality of sensitive and private user data is ensured when exchanged between a cloud application and an authorized user, and also when it is in storage. The cloud provider is conservatively considered largely untrusted. Scarce resources for mobile device users are conserved, by delegating responsibilities to the cloud provider, so that high scalability and economy can be achieved even in the context of a dynamic user population. The proposed solutions are not implementation-dependent and are applicable towards commercial cloud systems that are in operation today. In addition to key management, solutions have also been proposed for confidential information retrieval of public data from clouds. The significance of the research conducted is explained in Section 8.1; a summary of contributions, referencing each major work, is given in Section 8.2; finally, future directions of study for each are provided in Section 8.3.

8.1 Significance of Research

Although cloud computing systems have garnered significant interest from industry and the academic community from the perspective of their computation and storage capabilities, the area of security is still in early and active development; yet, security concerns have proved to be high barriers to adoption. The literature has yet to fully appreciate the unique challenges posed by a massively scalable cloud computing system, and how they invalidate traditional client-server encryption schemes. Solutions thus far have been unable to cope with the efficiency demands of a user population primarily composed of mobile devices, which is an evident trend in the marketplace. In a survey of work on secure mobile cloud computing, it was concluded that most security frameworks examined overlooked the tradeoff required between improved energy consumption on the mobile device, when offloading work to the cloud provider, and the increased expense of incurring communication with the cloud as a result of having to use the cloud's resources [63].

The work in this thesis proposes key management techniques that significantly reduce computation and communication costs for mobile users, while conservatively considering the cloud server to be untrusted. The mobile cloud applications being supported are forward-looking in that they are highly collaborative in nature, and depend upon data outsourcing and sharing; such applications differ from existing web-based systems that utilize a trusted server and entail one-to-one exchange of information. Furthermore, the thesis proposes how to deal effectively with additional important facets of cloud usage such as privacy of information retrieved from public databases in cloud systems. Many optional variants are presented throughout to support different cloud architectures, capabilities and classifications of users, as well as security attributes of data. In all cases, algorithms have been validated and benchmarked on popular smartphones and cloud systems in use today, which is often not the case with other existing works. It is imperative to conduct tests on actual device hardware, as performance factors such as throughput are difficult to simulate accurately, and for sound conclusions to be drawn as a result. Furthermore, scalability simulations have been run that extrapolate and help assess the viability of some of the techniques beyond a small testbed of devices. Thus, the thesis presents viable and realistic solutions for a significant amount of important use cases in mobile cloud computing.

8.2 Review of Contributions

The following contributions have been made in this thesis, with reference to the proposals and results detailed in previous chapters:

1. In Chapter 4, a novel solution is proposed that entails a key management scheme based on re-encryption that effectively utilizes the cloud for the most intensive cryptographic computation while preserving the confidentiality of data; other solutions rely upon a less scalable trusted third-party that acts as an intermediary in requests. Novel aspects are introduced to support a scalable and dynamic user population, such as a versioning array, key material sharing tactics by users, and intelligent timing of re-encryptions. A cloud-based prototype has been built to provide real world data and demonstrate the viability of the approach.

2. In Chapter 5, a protocol for outsourcing data storage to a cloud provider in secure fashion is provided. Authorized users qualify for access through possession of the right attributes without arbitration by the data owner. Unlike other attribute-based techniques, the protocol delegates computation and requests to a cloud provider or trusted authority. Responsibility over key generation is divided between a mobile data owner and a trusted authority; the owner is relieved of the highest computational burden. Additional security is provided through a group keying mechanism where the data owner controls access based on the distribution of an additional secret key; this additional security measure is an optional variant applicable to highly sensitive data subject to frequent access, resulting in a unique hybrid approach. Re-encryption permits efficient revocation of users; it does not require removal of attributes and subsequent key regeneration, and may be administered by a trusted authority without involvement of the data owner. Real-world benchmarks have been measured on a popular smartphone and cloud system, and calibrated simulations have been run to assess the scalability potential of the scheme.

3. In Chapter 6, the cloud's centralized data storage facility is used in novel fashion to store encryption key material as shares such that the provider cannot use them to decode user data also stored in the cloud. Unlike other key sharing techniques, the proposal makes use of the cloud's economical storage cost to maintain key material, and to degrade it over time, so that the cost of key re-generation is minimized. The

protocol uniquely exploits self-eroding key material in the cloud to achieve highly scalable access management for mobile users. The protocol has been implemented on a mobile cloud prototype, and a separate simulation program has demonstrated the effects of various key allocation and removal strategies in a realistic environment in which a large and dynamic mobile user population is modelled.

4. In Chapter 7, a novel hybrid technique is proposed that integrates location cloaking and private information retrieval in the context of a location-based service that leverages public data stored in the cloud. The proposal has been implemented on a client-server as well as mobile device hardware to determine its practicality in a resource-constrained environment. Users can achieve a good compromise between privacy and computational efficiency with the proposed technique unlike other existing location-based service proposals that seek to provide privacy guarantees. In particular, a trusted third-party is not needed to provide a cloaking function.

8.3 Future Work

Significant opportunities exist for advancing the research described in this thesis. Some suggested directions are presented, with reference to the work already described:

1. In the proposed scheme in Chapter 4, although the focus of the protocol is on data confidentiality, data integrity may be provided through the use of digital signaures or a similar mechanism to achieve a holistic security solution in case the cloud provider is deemed less trustworthy. Also, a hierarchical access control mechanism could be found to support different user classes and privileges. Furthermore, the appropriate timing of re-encryption activity in the cloud could be studied.

2. In the proposed scheme in Chapter 5, if a secret group key is utilized, it may be advantageous for a trusted manager to compute new key versions and re-encryption keys, and manage their storage and distribution. A suitable key versioning mechanism is suggested for this purpose, such as the one found in Chapter 4.

3. In the proposed scheme in Chapter 6, various additional heuristics for key share deletion may be explored, such as performing key re-generation functions in the

cloud during cheaper off-peak hours. Additionally, key shares may be deleted not based on the passage of time, but rather, based on the number of users that have left the authorized user set since the last deletion event; this practice may be more applicable for smaller populations.

4. In the proposed scheme in Chapter 7, the general scenario where the user retrieves all of the POIs that belong to the VHC cell of interest could be modified. The user could be allowed expand the search for POIs by searching in a broader geographical area through an additional query, or the user could request POIs for all of the VHC cells within a geographic grid cell to obtain useful results. Applicability of the scheme to other application domains such as multimedia content could also be studied.

Although the use of data encryption appears to counteract the economic advantages of running applications in an open and scalable manner in the cloud, it is a concession made to prevent an untrusted cloud provider from learning any confidential information. Operations by cloud applications on encrypted data, such as indexing and searching, is an open research problem. Fully homomorphic encryption schemes have been proposed that seemingly make this possible [43]; although they are still largely impractical, they show promise. Furthermore, there is recent work on practical means of searching through encrypted data stored in clouds based on fuzzy keyword search techniques [72, 95].

Finally, it would be interesting to apply the techniques proposed in this thesis to various real-world mobile cloud applications to gauge their performance on realistic workloads and user populations.

APPENDICES

Appendix A

Mobile and Cloud Computing Costs

Introduction

The techniques presented in this thesis strive to minimize energy consumption for mobile device users; the importance of doing so is demonstrated in Section A.1. Another goal is conserving computation on a cloud computing server; although it is scalable, its computational workload may be substantial and incur expenses for the client, as shown in Section A.2. This appendix provides justification for optimizing these factors.

A.1 Mobile Device Energy Consumption

A useful consideration in security cost estimation is energy usage from the exchange of communications. Smartphones consume considerable energy when transmitting and receiving, as compared to their idle states. In the case of a mobile device operating in the 3G network mode, current draw is approximately 100 mA in idle state, as measured on an HP iPAQ smartphone [99]. The same study also found that when transmitting, current draw peaks at approximately 300 mA on ramp-up, with an energy tail of approximately an average of 200 mA sustained for an additional 16 seconds, resulting in a considerable aggregate drain. This is due to the fact that a 3G wireless radio is maintained in high-power active state by the network to maximize responsiveness and minimize the signalling costs of additional transmissions. Similarly, another study of energy consumption was conducted on Nokia

N95 phones, capable of HSDPA/UMTS and also operating in 3G mode [11]. It was found that nearly 60% of energy, the so-called *tail energy*, is wasted in high-power states after completing a transfer, while the initial ramp energy is small; the design motivation is to reduce ramp-up delay in subsequent transfers. In comparison, the older and slower GSM network operating mode is characterized by a tail time that is half that of the 3G mode. Uploads were found to consume more energy than downloads; for example, the transfer energy for uploads is nearly 30% greater, for 100 KB transfers [11].

Equation A.1 describes the energy consumption in a single upload request from a 3G mobile device:

$$E_{total} = R \cdot \left[\frac{S(M)}{T} \cdot t_{peak} \cdot C_{peak} + t_{tail} \cdot C_{tail} \right] \tag{A.1}$$

where R is the number of requests, $S(M)$ is the size of each wireless request, T is the throughput, t_{peak} and t_{tail} are the ramp-up and high-power state durations, respectively, and C_{peak} and C_{tail} are the peak and tail current draws, respectively. Figure A.1 illustrates the components of energy consumption described.

Consider an example of the energy consumption of a mobile device with a typical throughput of 1200 kbps on an HSDPA 3G network; this rate represents a realistic throughput as determined in a mobile driving test of HSDPA downloads on a real network, performed by Ericsson [34, 99]. Assume an average capacity of 1200 mAh for a typical smartphone battery. For instance, the Apple iPhone 3GS includes an internal battery with a capacity of 1219 mAh. Therefore, total battery consumption in mAh for a transfer request is as follows:

$$E_{total} = R \cdot \left[\frac{S(M) \cdot 300mA}{1200\frac{kb}{s} \cdot 3600\frac{s}{h}} + \frac{16s \cdot 200mA}{3600\frac{s}{h}} \right]$$

$$E_{total} = R \cdot \left[S(M) \cdot 6.94 \cdot 10^{-5} + 8.89 \cdot 10^{-1} \right] mAh$$

If the user initiates 100 requests of 1 MB each per day, such that:

$$R = 100, S(M) = 1MB$$

then the daily energy consumption is:

$$E_{total} = 100 \cdot \left[1 \cdot 10^{3} \cdot 6.94 \cdot 10^{-5} + 8.89 \cdot 10^{-1} \right] mAh = 95.83mAh = 345C$$

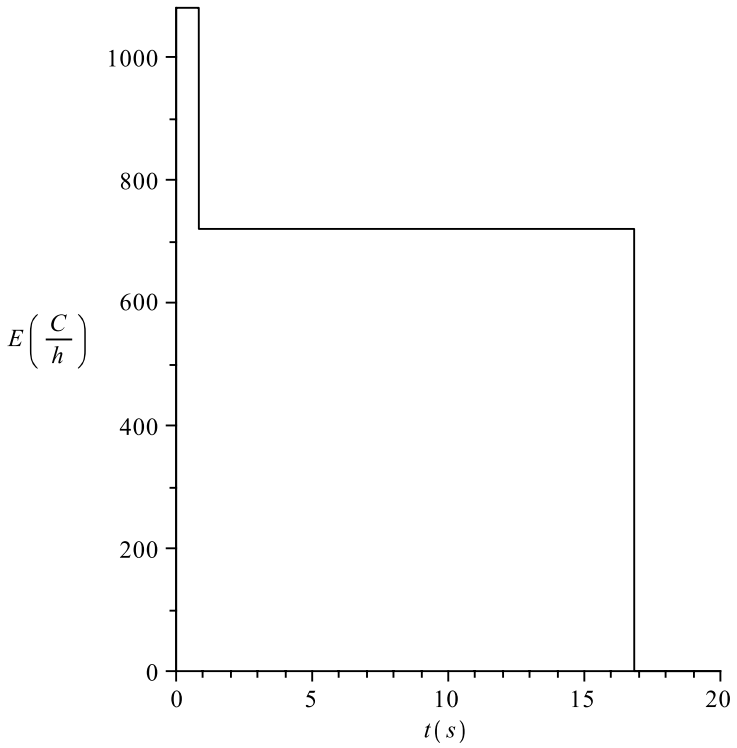


Figure A.1: The energy consumption of a single transmission on a mobile device, in $\frac{C}{h}$ over a span of time (where $3.6C = 1mAh$), illustrating the initial energy ramp-up during packet transmission, followed by the tail of the high-energy state.

If this energy consumption is doubled to account for the receipt of responses or messages from the cloud, then the total resultant energy consumption for radio communication alone represents a significant 16.0% of a typical smartphone battery; this figure does not include the energy cost of processing messages and storing them to flash memory. Given these characteristics of energy consumption, it is advantageous to limit the number of small transfers on mobile phones, each of which incur high overhead, in any security protocol.

In [11], it is suggested that scheduling changes can occur. For instance, transmissions may be scheduled together in delay-tolerant applications so that the time spent in the high power state is minimized. In addition, applications may prefetch data to minimize the total number of transactions. Reduction of message traffic is a key goal in this work.

A.2 Cloud Server Cryptographic Workload

Another factor in cost estimation is the amount of cryptographic processing. To determine the workload on the server for a cloud application serving a large user population, assume a homogeneous stochastic Poisson process defined in Equation A.2:

$$P\left[(N(t+\tau)-N(t))=k\right]=\frac{(\lambda\tau)^k e^{-\lambda\tau}}{k!} \quad \text{for} \quad k=0,1,..., \tag{A.2}$$

where λ is the expected number of elements, k is the number of events in the time interval $(t, t+\tau]$ and λ is the expected number of elements per unit time. In the context of this study, the elements are the number of users of the cloud application, and the events are the requests that they initiate against the cloud application.

Consider an example of a workload calculation for an authentication server deployed in the cloud; assume appropriate values for the parameters in the cloud usage scenario as found in Table A.1. The usage frequency is hypothetical and intended to resemble an active cloud application user. The encryption rates are derived from published results of Crypto++, an open-source cryptographic library [31]. Benchmark tests were conducted on random data blocks on an Intel dual-core 1.83 GHz desktop, using the Microsoft Vista OS. The RSA operations are based on a key length of 2048 bits, and it is assumed that blocks of 245 bytes (256 bytes less 11 for user data) are encrypted at a time, using PKSC #1 padding. The AES operations are based on CBC (Cipher Block Chaining) mode. The tests omit memory operations; it is expected that slower times would be achieved in practice.

Symbol	Definition	Assumed values in large-scale system
U	Users accessing authentication server	1 million
	Accesses per user per day	10
λ	Expected event rate per server	1 million requests per day
τ	Time interval (maximum wait tolerated by mobile user)	10 s
m_{size}	Average message size	1 MB
$r_{enc\ RSA}$	Encryption rate using RSA	0.0787 s per MB
$r_{enc\ AES}$	Encryption rate using AES	0.0092 s per MB
$r_{dec\ RSA}$	Decryption rate using RSA	2.9921 s per MB
$r_{dec\ AES}$	Decryption rate using AES	0.0092 s per MB
μ	Service rate	0.5

Table A.1: The parameters for the arrivals in a cloud application.

Suppose that a single authentication server can handle up to $\left\lfloor \frac{60s}{r_{enc} \text{ or } r_{dec}} \cdot \mu \right\rfloor$ requests per minute, assuming a maximum sustained server utilization of 0.5, to maintain responsiveness. Choose an interval with a duration of 10 seconds, which is assumed to be the maximum delay tolerated by a mobile user. With an average sustainable performance of approximately 2 $\frac{s}{MB}$ on a cryptographic operation, this equates to a rate of 2.5 requests per interval.

Assume a global population of users U accessing a set of authentication servers A with frequency λ, further assuming an even distribution of requests. Given Equation A.2, the estimated probability of exceeding an authentication server's capacity is:

$$\begin{aligned} P(k > 2, \tau) &= 97.9\% \quad \text{for 2000 servers,} \\ &= 88.9\% \quad \text{for 1000 servers,} \\ &= 59.2\% \quad \text{for 500 servers} \end{aligned}$$

The probability mass function is shown in Figure A.2. Thus, approximately one authentication server per 500 users is required to handle cryptographic operations for the entire user population. With a more conservative allocation, the cloud will be unable to meet the desired throughput of user requests.

If the assumption about an evenly distributed workload is modified, and it is discovered

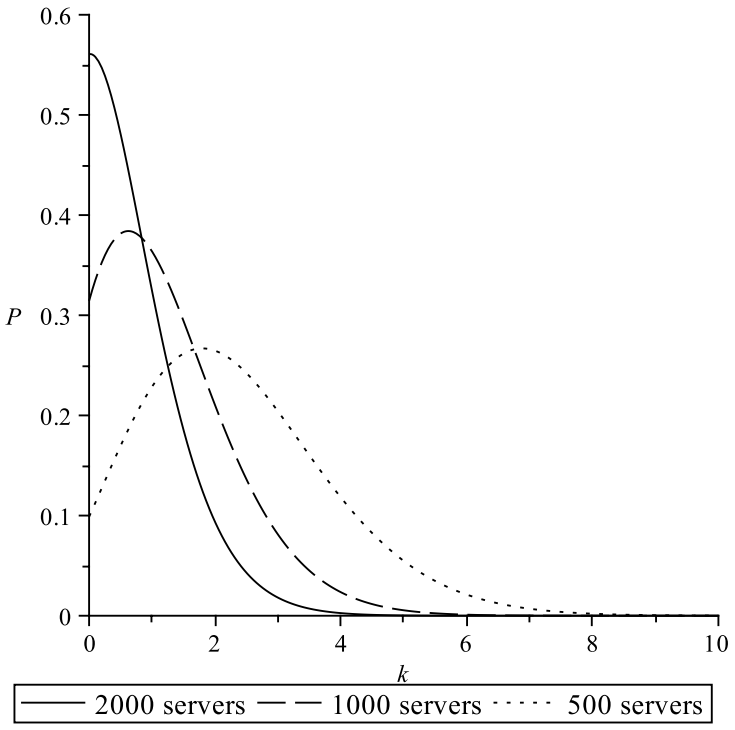


Figure A.2: A cloud server arrival model, where a Poisson probability mass function shows the probability of a user request against an authentication server within an acceptable delay interval.

that requests substantially increase at peak times, then even greater capacity in the authentication centre will be needed at additional expense. In fact, it has been demonstrated that to achieve a high degree of on-demand availability in a cloud computing system, at least *twice* the number of nodes in the largest collection of nodes under study in the system (such as the servers responsible for authentication in this example), must be made available during a busy period [52]; this analysis is based on the modelling, using a classic Erlang loss model, of multiple resource classes in a cloud; these classes contain different numbers of computational nodes that can be assigned to user tasks in a cloud system. Recall that in cloud computing, a user will pay for the use of different classes of resources to achieve a compromise between total processing time and cost. If such an allocation of servers cannot be achieved, then the detrimental effect of *blocking* can occur; an incoming authentication request from a user will be denied or simply dropped. Note that this behaviour is unlike that of a grid computing system, in which immediate access to the system's computational resources are not generally expected, and therefore requests may be queued during times of peak processing.

There is little question that authentication operations incur a considerable computational penalty on a server. In one test, Microsoft measured the performance implications of client authentication on ProLiant web servers [35]. The throughput, in responses per second given a 50 user load, was approximately 10 times worse when using basic SSL authentication over non-authenticated anonymous requests. As well, the response time was approximately four times worse.

Rather than attempting to expand the capacity of the authentication centre to achieve a cloud system with highly scalable and available security, the option exists to reduce the number of cryptographic requests through a different user authentication and data encryption model, such as a co-operative one that off-loads cryptographic operations from the cloud to a trusted manager entity at increased but manageable cost to the user.

References

[1] "Security Guidance for Critical Areas of Focus in Cloud Computing V2.1," Cloud Security Alliance, Tech. Rep., December 2009.

[2] "Oracle Enterprise Transformation Solutions Series: Cloud Reference Architecture," Oracle, Tech. Rep., November 2012.

[3] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," Cryptology ePrint Archive, Report 2003/126, 2003, http://eprint.iacr.org/.

[4] H. S.-M. Ali Khoshgozaran and C. Shahabi, "SPIRAL, a scalable private information retrieval approach to location privacy," in *Proceedings of the 2nd International Workshop on Privacy-Aware Location-based Mobile Services (PALMS)*, 2008.

[5] Amazon, "Amazon Web Services: Overview of Security Processes," November 2009. [Online]. Available: http://aws.amazon.com/security

[6] ——. (2012) Amazon S3 Pricing. [Online]. Available: http://aws.amazon.com/s3/pricing/

[7] M. J. Atallah, K. B. Frikken, and M. Blanton, "Dynamic and efficient key management for access hierarchies," in *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 2005, pp. 190–202.

[8] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions of Information and System Security*, vol. 9, pp. 1–30, Feb. 2006.

[9] J. Baek, J. Newmarch, R. Safavi-naini, and W. Susilo, "A survey of identity-based cryptography," in *Proc. of Australian Unix Users Group Annual Conference*, 2004, pp. 95–102.

[10] J. Baek and Y. Zheng, "Identity-Based Threshold Decryption," in *Public Key Cryptography – PKC 2004*, ser. Lecture Notes in Computer Science, F. Bao, R. Deng, and J. Zhou, Eds. Springer Berlin / Heidelberg, 2004, vol. 2947, pp. 262–276.

[11] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293.

[12] B. Bamba, L. Liu, P. Pesti, and T. Wang, "Supporting anonymous location queries in mobile environments with privacygrid," in *Proceeding of the 17th international conference on World Wide Web*, New York, NY, USA, 2008, pp. 237–246.

[13] Barker, E., et al, "Recommendation for key management," NIST, Tech. Rep. NIST Special Publication 800-57, March 2007.

[14] M. Behrendt, B. Glasner, P. Kopp, R. Dieckmann, G. Breiter, S. Pappe, H. Kreger, and A. Arsanjani, "Introduction and Architecture Overview: IBM Cloud Computing Reference Architecture 2.0," IBM, Tech. Rep., 2011.

[15] A. Beimel and Y. Stahl, "Robust information-theoretic private information retrieval," *J. Cryptol.*, vol. 20, no. 3, pp. 295–321, 2007.

[16] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: dependable and secure storage in a cloud-of-clouds," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 31–46.

[17] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 321–334.

[18] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *In EUROCRYPT*. Springer-Verlag, 1998, pp. 127–144.

[19] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology — CRYPTO 2001*, ser. Lecture Notes in Computer Science, J. Kilian, Ed. Springer Berlin / Heidelberg, 2001, vol. 2139, pp. 213–229.

[20] J. Brodkin, "Gartner: Seven Cloud-Computing Security Risks," *Network World*, July 2008.

[21] W. E. Burr, D. F. Dodson, E. M. Newton, R. A. Perlner, W. T. Polk, S. Gupta, and E. A. Nabbus, "Electronic Authentication Guideline," National Institute of Standards and Technology (NIST), Tech. Rep. Special Publication 800-63-1, December 2011.

[22] A. Chakrabarti, *Grid Computing Security*. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 2007.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *OSDI*, 2006, pp. 205–218.

[24] D. Chappell, "Introducing the Windows Azure Platform," Microsoft, Tech. Rep., 2009.

[25] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of the 36th Annual Symposium on the Foundations of Computer Science, 1995*, Oct 1995, pp. 41–50.

[26] B. Chor and N. Gilboa, "Computationally private information retrieval (extended abstract)," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, 1997, pp. 304–313.

[27] B. Chor, N. Gilboa, and M. Naor, "Private information retrieval by keywords," Dept. of Computer Science, Technion, Israel, Tech. Rep. TR CS0917, 1997.

[28] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[29] C. Chow, M. F. Mokbel, and X. Liu, "A peer-to-peer spatial cloaking algorithm for anonymous location-based service," in *Proceedings of the 14th Annual ACM international Symposium on Advances in Geographic information Systems*, New York, NY, USA, 2006, pp. 171–178.

[30] R. Chow, M. Jakobsson, Y. Niu, E. Shi, J. Molina, R. Masuoka, and Z. Song, "Authentication in the clouds: a framework and its application to mobile users," in *ACM Cloud Computing Security Workshop (CCSW)*, October 8, 2010 2010.

[31] W. Dai, "Crypto++ 5.6.0 benchmarks," March 2009. [Online]. Available: http://www.cryptopp.com/benchmarks.html

[32] Y.-S. Dai, X. Zou, and Y. Pan, *Trust and Security in Collaborative Computing*. World Scientific, 2007, vol. Volume 2 of Computer and Network Security.

[33] A. De Caro, "Java Pairing-Based Cryptography Library," 2012. [Online]. Available: http://libeccio.dia.unisa.it/projects/jpbc/

[34] J. Derksen, R. Jansen, M. Maijala, and E. Westerberg, "HSDPA Performance and Evolution," *Ericsson Review*, vol. No. 3, p. 117, 2006.

[35] P. Dhawan, "Performance comparison: Security design choices," Microsoft Developer Network, Tech. Rep., October 2002.

[36] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2004, pp. 21–21.

[37] J.-M. Do, Y.-J. Song, and N. Park, "Attribute based proxy re-encryption for data confidentiality in cloud computing environments," in *Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference on*, may 2011, pp. 248 –251.

[38] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[39] enStratus Networks, "enStratus Security Architecture: Design of the enStratus System and General enStratus Security Policies," May 2010.

[40] Facebook, "Statistics," 2010. [Online]. Available: "http://www.facebook.com/press/info.php?statistics"

[41] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data," in *Proc. of the 18th USENIX Security Symposium*, 2009.

[42] F. Gens, "IDC's New IT Cloud Services Forecast: 2009-2013," October 5, 2009 2009. [Online]. Available: http://blogs.idc.com/ie/

[43] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[44] G. Ghinita, "Understanding the privacy-efficiency trade-off in location based queries," in *SPRINGL '08: Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*, New York, NY, USA, 2008, pp. 1–5.

[45] G. Ghinita, P. Kalnis, M. Kantarcioglu, and E. Bertino, "A hybrid technique for private location-based queries with database protection," in *SSTD '09: Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 98–116.

[46] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2008, pp. 121–132.

[47] I. Goldberg, "Percy++ project on SourceForge," http://percy.sourceforge.net/.

[48] ——, "Improving the robustness of private information retrieval," in *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2007, pp. 131–148.

[49] Google, "Google App Engine," November 2009. [Online]. Available: http://code.google.com/appengine/

[50] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference*

on Computer and Communications Security, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 89–98.

[51] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, New York, NY, USA, 2003, pp. 31–42.

[52] T. J. Hacker, *Cloud Computing and Software Services*. CRC Press, 2011, ch. Toward a Reliable Cloud Computing Service, pp. 139–152.

[53] U. Hengartner, "Hiding location information from location-based services," in *Mobile Data Management, 2007 International Conference on*, May 2007, pp. 268–272.

[54] J. Hur and D. K. Noh, "Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1214–1221, 2011.

[55] IDC, "Press Release: Worldwide Converged Mobile Device (Smartphone) Market Grows 56.7% Year Over Year in First Quarter of 2010," May 7 2010.

[56] A. Iliev and S. W. Smith, "Protecting Client Privacy with Trusted Computing at the Server," *IEEE Security and Privacy*, vol. 3, no. 2, pp. 20–28, 2005.

[57] S. Jahid, P. Mittal, and N. Borisov, "EASiER: encryption-based access control in social networks with efficient revocation," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11. New York, NY, USA: ACM, 2011, pp. 411–415.

[58] E. jin Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *in Proc. Network and Distributed Systems Security (NDSS) Symposium 2003*, 2003, pp. 131–145.

[59] Juniper Research, "Mobile Cloud Applications & Services: Monetising Enterprise & Consumer Markets 2009-2014," Juniper Research, Tech. Rep., 2010.

[60] Kallahalla, M., et al, "Plutus: Scalable secure file sharing on untrusted storage," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2003, pp. 29–42.

[61] S. Kamara, C. Papamanthou, and T. Roeder, "CS2: A Searchable Cryptographic Cloud Storage System," Microsoft Research, Tech. Rep. MSR-TR-2011-58, May 2011.

[62] M. Kennedy and S. Kopp, *Understanding Map Projections*. ESRI (Environmental Systems Research Institute) press, 2000.

[63] A. N. Khan, M. L. M. Kiaha, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.08.003

[64] Kim, Y., et al, "Key establishment scheme for sensor networks with low communication cost," in *Autonomic and Trusted Computing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4610, pp. 441–448.

[65] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, ser. Lecture Notes in Computer Science, vol. 773. Springer, 1993, pp. 136–146.

[66] R. L. Krutz and R. D. Vines, *"Cloud Security: A Comprehensive Guide to Secure Cloud Computing"*. Wiley Publishing, 2010.

[67] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval," in *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1997, p. 364.

[68] N. Leavitt, "Is Cloud Computing Really Ready for Prime Time?" *Computer*, vol. 42, pp. 15–20, January 2009.

[69] T. Leighton, "White Paper: Akamai and Cloud Computing: A Perspective from the Edge of the Cloud," Akamai, Tech. Rep., 2009.

[70] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's Inside the Cloud? An Architectural Map of the Cloud Landscape," in *CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–31.

[71] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-based authentication for cloud computing," in *Cloud Computing*, ser. Lecture Notes in Computer Science, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin / Heidelberg, 2009, vol. 5931, pp. 157–166.

[72] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 441–445.

[73] X. Liang, R. Lu, and X. Lin, "Ciphertext policy attribute based encryption with efficient revocation," University of Waterloo, Technical Report BBCR, 2011.

[74] Q. Liu, G. Wang, and J. Wu, "Clock-based proxy re-encryption scheme in unreliable clouds," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, sept. 2012, pp. 304 –305.

[75] C. Lynch and F. O. Reilly, "Processor choice for wireless sensor networks," in *REALWSN'05: Workshop on Real-World Wireless Sensor Networks*, 2005, pp. 1–5.

[76] B. Lynn, "PBC (Pairing-Based Cryptography) Library," 2012. [Online]. Available: http://crypto.stanford.edu/pbc/

[77] T. Mather, "Key Management in the Cloud," January 2010. [Online]. Available: https://365.rsaconference.com/blogs/tim-mather/2010/01/07/key-management-in-the-cloud

[78] T. Mather, S. Kumaraswamy, and S. Latif, "Cloud Security and Privacy." O'Reilly, 2009.

[79] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," October 2009.

[80] J. V. D. Merwe, D. Dawoud, and S. McDonald, "A Survey on Peer-to-Peer Key Management for Mobile Ad Hoc Networks," *ACM Comput.Surv.*, vol. 39, no. 1, p. 1, 2007.

[81] Microsoft, "Press Release: Microsoft and Living PlanIT Partner to Deliver Smart City Technology Via the Cloud," March 2011.

[82] Y. Ming, L. Fan, H. Jing-Li, and W. Zhao-Li, "An efficient attribute based encryption scheme with revocation for outsourced data sharing control," in *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, 2011, pp. 516–520.

[83] S. K. Mishra and P. Sarkar, "Symmetrically private information retrieval," in *INDOCRYPT '00: Proceedings of the First International Conference on Progress in Cryptology*, London, UK, 2000, pp. 225–236.

[84] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new Casper: query processing for location services without compromising privacy," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 763–774.

[85] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, New York, NY, USA, 1999, pp. 245–254.

[86] K. Nohl and C. Paget, "Gsm: Srsly?" Presentation at 26th Chaos Communication Congress, Berlin, Dec. 2009.

[87] OASIS, "Key Management Interoperability Protocol (KMIP): Addressing the Need for Standardization in Enterprise Key Management," 2009.

[88] F. Olumofin, P. K. Tysowski, I. Goldberg, and U. Hengartner, "Achieving Efficient Query Privacy for Location Based Services," Centre for Applied Cryptographic Research (CACR), University of Waterloo, Tech. Rep. 22, 2009.

[89] ——, "Achieving Efficient Query Privacy for Location Based Services," in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 93–110.

[90] S. Pearson, "Taking Account of Privacy when Designing Cloud Computing Services," in *CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–52.

[91] A. Pingley, W. Yu, N. Zhang, X. Fu, and W. Zhao, "CAP: A Context-Aware Privacy Protection System For Location-Based Services," in *29th IEEE International Conference on Distributed Computing Systems*, Jun 2009.

[92] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, "Enabling Security in Cloud Storage SLAs with CloudProof," in *USENIX Annual Technical Conference*, 2011.

[93] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 85–100.

[94] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *Internet Computing, IEEE*, vol. 16, no. 1, pp. 69–73, 2012.

[95] ——, "Toward secure and effective data utilization in public cloud," vol. 26, no. 6, 2012, pp. 69–74.

[96] F. Saint-Jean, "Java implementation of a single-database computationally symmetric private information retrieval (CSPIR) protocol," Yale University, New Haven, CT, USA, Technical Report YALEU/DCS/TR-1333A, 2005.

[97] B. Schneier, *Applied Cryptography: Protocols, algorithms, and source code in C*. Wiley, 1994.

[98] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[99] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding, "Cool-Tether: Energy Efficient On-the-fly WiFi Hot-spots using Mobile Phones," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*. New York, USA: ACM, 2009, pp. 109–120.

[100] Y. Shin, M. Gupta, and S. Myers, "A Study of the Performance of SSL on PDAs," in *Proceedings of IEEE INFOCOM Global Internet Symposium (GI)*, 2009, pp. 1–6.

[101] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 3rd ed. McGraw-Hill, 1999.

[102] R. Sion and B. Carbunar, "On the computational practicality of private information retrieval," in *Proceedings of the Network and Distributed Systems Security Symposium*, 2007.

[103] J. P. Snyder, *Flattening the Earth, two thousand years of map projections*. University of Chicago Press, 1993.

[104] A. Solanas, J. Domingo-Ferrer, and A. Martínez-Ballesté, "Location privacy in location-based services: Beyond TTP-based schemes," in *PiLBA*, ser. CEUR Workshop Proceedings, C. Bettini, S. Jajodia, P. Samarati, and X. S. Wang, Eds., vol. 397, 2008.

[105] A. Stern, "Update From Amazon Regarding Friday's S3 Downtime," Febuary 16, 2008 2008. [Online]. Available: http://www.centernetworks.com/amazon-s3-downtime-update

[106] A. Tassanaviboon and G. Gong, "OAuth and ABE based authorization in semi-trusted cloud computing: aauth," in *Proceedings of the second international workshop on Data intensive computing in the clouds*, ser. DataCloud-SC '11. New York, NY, USA: ACM, 2011, pp. 41–50.

[107] T. Tiemens. (2012, May) Shamir Secret Sharing in Java. [Online]. Available: http://sourceforge.net/projects/secretsharejava/

[108] P. K. Tysowski and M. A. Hasan, "Re-Encryption-Based Key Management Towards Secure and Scalable Mobile Applications in Clouds," Cryptology ePrint Archive, Tech. Rep. 668, 2011.

[109] ——, "Towards Secure Communication for Highly Scalable Mobile Applications in Cloud Computing Systems," Centre for Applied Cryptographic Research (CACR), University of Waterloo, Tech. Rep. 33, 2011.

[110] ——, "Cloud-Hosted Key Sharing Towards Secure and Scalable Mobile Applications in Clouds," in *2nd International Conference on Computing, Networking and Communications (ICNC)*, 2013.

[111] ——, "Hybrid Attribute-Based Encryption and Re-Encryption for Scalable Mobile Applications in Clouds," Centre for Applied Cryptographic Research (CACR), University of Waterloo, Tech. Rep. 13, 2013.

[112] P. K. Tysowski, P. Zhao, and K. Naik, "Peer To Peer Content Sharing on Ad Hoc Networks of Smartphones," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, July 2011, pp. 1445 –1450.

[113] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 735–737.

[114] J. Wang, "Java Realization for Ciphertext-Policy Attribute-Based Encryption," 2012. [Online]. Available: http://github.com/wakemecn

[115] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, ser. StorageSS '08. New York, NY, USA: ACM, 2008, pp. 21–26.

[116] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 16–30, February 2000.

[117] T. Xu and Y. Cai, "Location anonymity in continuous location-based services," in *Proceedings of the 15th Annual ACM international Symposium on Advances in Geographic information Systems*, New York, NY, USA, 2007, pp. 1–8.

[118] L. Yan, C. Rong, and G. Zhao, "Strengthen Cloud Computing Security with Federal Identity Management Using Hierarchical Identity-Based Cryptography," in *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 167–177.

[119] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 534–542.

[120] ——, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 261–270.

[121] L. Zeng, Z. Shi, S. Xu, and D. Feng, "Safevanish: An improved data self-destruction for protecting data privacy," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 521–528.

[122] M. Zhang, C. Carroll, and A. Chan, "Analysis of is-95 cdma voice privacy," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, D. Stinson and S. Tavares, Eds. Springer Berlin / Heidelberg, 2001, vol. 2012, pp. 1–13.

[123] G. Zhao, C. Rong, J. Li, F. Zhang, and Y. Tang, "Trusted data sharing over untrusted cloud storage providers," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 97–103.

[124] G. Zhong and U. Hengartner, "A distributed k-anonymity protocol for location privacy," in *Proceedings of Seventh IEEE International Conference on Pervasive Computing and Communication (PerCom 2009). Galveston, TX*, 2009, pp. 253–262.

[125] P. Zimmermann, "Pretty good privacy: public key encryption for the masses," in *Building in big brother*, L. J. Hoffman, Ed. New York, NY, USA: Springer-Verlag New York, Inc., 1995, pp. 93–107.