Towards Secure Communication for Highly Scalable Mobile Applications in Cloud Computing Systems

Piotr K. Tysowski and M. Anwarul Hasan
Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
pktysowski@uwaterloo.ca and ahasan@uwaterloo.ca

Abstract

Cloud computing is a distributed computing model in which clients pay for computing and data storage resources of a third-party cloud provider. Cloud applications can scale up or down almost instantly to meet real-time computational demands. Despite the economic advantages of a pay-on-demand business model, security remains a paramount concern. Cloud applications by nature must store and retain access to a potentially large volume of data. Yet, the consensus among IT organizations is that the cloud provider generally cannot be implicitly trusted, and thus this data should ideally be transmitted and stored in encrypted form. Major challenges exist concerning the aspects of the generation, distribution, and usage of encryption keys in cloud systems, such as the safe location of keys, and the presence of users that tend to connect to contemporary cloud applications using resource-constrained mobile devices in extremely large numbers simultaneously; these characteristics lead to difficulties in achieving efficient and highly scalable key management. In this work, at first, leading models for key distribution are applied to a cloud computing system. The underlying schemes include identity-based encryption, multi-level key management, and data re-encryption. Each model varies in how user keys are stored, exchanged, re-generated, and used; each is analyzed and compared in terms of storage and communication requirements, assessing the impact on tight resource constraints in a mobile device environment; these constraints include costly wireless data usage, limited mobile processing power, and limited battery life. The analysis is applicable in light of current trends towards mobile devices communicating with interactive cloud applications. Next, a model for key distribution based on the principle of dynamic data re-encryption is modified and applied to a cloud computing system in a unique way. The proposed cloud-based re-encryption model is secure, efficient, and scalable in a cloud computing context, as keys are managed by the client for trust reasons, processor-intensive data re-encryption is handled by the cloud provider, and key redistribution is minimized to conserve communication on mobile devices. A versioning history mechanism effectively manages keys for a continuously changing user population. Finally, an implementation on commercial mobile and cloud platforms is used to validate the performance of the models.

1 Introduction

Cloud computing is an evolutionary new model for distributed computing consisting of centralized data centres that provide resources for massively scalable units of computing. These computational facilities are delivered as a service to users over an insecure medium such as the Internet, and may be bridged to wireless packet data networks. A client of a cloud provider can address changes in

demand for its processing needs by replicating applications in the cloud to many runtime instances, and running them on cloud servers in concurrent fashion. Unanticipated burst demands such as flash traffic on a web server may be met automatically without noticeable delay. The client does not need to incur a high capital expense up front in anticipation of future application usage patterns that may be difficult to predict accurately, and could otherwise lead to outages if left unaddressed; excess capacity and idle cycles are avoided. The easy scalability of cloud applications, in light of access to nearly unlimited computing resources, results in equal opportunity of benefits to firms large and small.

Yet, despite all its economic benefits, the cloud computing model poses very significant risks to its users. Because user data is stored and executed within the domain of the cloud, and there is little or no visibility into how the cloud is implemented and internally managed by the cloud provider, there is significant concern over the security and privacy of transactions and data permanently stored in the cloud. Security tends to be enforced by the provider, not the client. Dominant opinion is that data ought to be kept confidential not only from other users sharing the cloud, but also from the cloud provider itself, as much as possible. Indeed, a survey of Chief Information Officers and IT executives by IDC rated security as the chief concern in the use of cloud computing services [1]; the concern is that the client requests storage of application logic and data in the cloud without assurance of exactly where it is stored, whether it is replicated or cached, how long it is kept for, and who exactly has access to it.

Despite its need for protection, cloud data must remain highly accessible. Large multi-user cloud applications such as e-collaboration networks must be efficiently accessed by a heterogeneous mix of computing devices, while the cost of communication must be minimized for mobile device users. After all, such costs are borne by the client. The goal of security researchers is to develop techniques to ensure communication security in cloud computing systems at reasonable cost. Only by overcoming these challenges, will enterprise companies invest in and migrate to the cloud to reap its economic benefits. The topic of this work is the adaptation of leading key management schemes, that were originally designed for a traditional client-server context, in a novel way that addresses the communication security challenges of the cloud environment. The intent is to find viable ways to protect the security of the communication between the cloud and its users, as well as to protect the privacy of the stored cloud data, in an efficient and highly scalable way.

In this work, a system model comprising a mobile-based cloud application is first presented. A threat model points out possible attacks that must be prevented in order to possess a secure cloud framework. Related work in cloud communication security follows. Three leading key management models for a cloud computing system are proposed, with differing design philosophies reflected in the choices; the underlying techniques are based on existing proposals but have been adapted here to a mobile-based cloud computing system, with appropriate modifications to key storage locations and distribution strategies, and to workflows. These schemes include key management through a centralized cloud service, through a trusted client-side authority, and through the use of automatic data re-encryption performed outside the cloud. Finally, a novel scheme is presented where the latter model is further modified so that the re-encryption workload is processed by the cloud provider as an untrusted entity. Implementation results on popular smartphone and cloud platforms are provided that validate the assumptions made. Finally, concluding remarks on the efficiency and scalability of the proposed schemes are made.

Contributions of this work include the framing of the problem of achieving very high scalability in a mobile-based cloud application and the adaptation of leading key management techniques from older and unrelated systems to a contemporary cloud computing application that serves a mobile user population and brings its own unique challenges, as well as a comparative analysis of the impact of the schemes in terms of efficiency, scalability, and trust. Additionally, a novel

solution is proposed, which entails a key management scheme based on re-encryption that effectively utilizes the cloud for cryptographic computation while supporting a frequently-changing mobile user population that does not need to trust the cloud provider; novel aspects such as a versioning array, key material sharing tactics by users, and intelligent timing of re-encryptions, make it possible. A cloud-based prototype has also been built to provide real world data and demonstrate the viability of the approach. This work is the only one that the authors are aware of that provides a secure communication solution for a forward-looking cloud system accessed by potentially millions of resource-constrained device users.

2 System model

A contemporary system model is adopted here. The system under study is a *public cloud* provider, such as the Amazon Elastic Compute Cloud (EC2) or Google App Engine (GAE), operating a centralized data centre that is accessed by a large mobile user population over an unprotected public Internet network infrastructure bridged to a wireless network. A user may access the cloud application from a mobile device such as a tablet, smartphone, or even a wireless sensor. A highly scalable multi-user cloud application is envisioned; it may service a large user population for the purpose of e-collaboration, a social network, or customer relationship management. It is continuously accessed by a multitude of heterogeneous mobile device users. Other possibilities for cloud access include an in-vehicle telematics solution delivering real-time traffic information from a central database over a dedicated 802.11p network, and an environmental sensor reporting to an analytics back-end over a ZigBee wireless link. Each mobile device user typically opens a direct TCP/IP connection to a cloud application portal. The distinction between a mobile user and a fixed one is important from the standpoint of the cloud provider for various reasons: in the wireless case, communication sessions with users must be minimized to avoid unnecessary energy drain and to avoid incurring high wireless data usage costs; expensive client-side calculations cannot be expected; and, the availability of mobile devices may be limited due to connectivity loss.

Users may upload and download content to and from the cloud by having the cloud-hosted front end access a data partition provisioned by a cloud's blade server. Each data partition is made accessible to a set of authorized mobile users. The client organization is responsible for granting access rights. The cloud provider is not fully trusted, and although it may assist in enforcing those rights for users, it cannot gain access to the data itself. This notion is in keeping with Gartner's notion of cloud computing security implying a shared environment in which data is segregated and encrypted [2]. In some variations of the encryption models presented, mention is made of a *private cloud*, where the efficiencies of cloud computing are realized on an infrastructure that is 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, although it is generally less economical, overall. In a private cloud, a client may access the server over a protected intranet, behind an external firewall, or as a Virtual Private Network (VPN) accessed via a secure tunnel.

Robustness of the cloud provider is not a significant concern in this study, as 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. Any role the cloud provider plays in executing a security scheme is assumed to impart the same level of confidence as its other services and does not really constitute a single point of failure. The assumptions driving the view of a highly scalable mobile cloud application are further elaborated below:

1. *User mobility.* It is evident that user-driven cloud computing applications are being primarily

accessed by mobile device users. There has been a clear trend towards adoption of smartphone devices; approximately 54.7 million smartphones were sold in the first quarter alone of 2010, up 56.7% from the same quarter a year ago [3]. Correspondingly, the market for cloud-based mobile applications is expected to grow at an annual average rate of 88% from 2009 to 2014. The market was just over $400 million in 2009, according to Juniper, but by 2014 it is expected to reach $9.5 billion. Driving this growth will be technologies such as the new HTML 5 web standard, increased mobile broadband coverage, and the need for always-on collaborative services for the enterprise [4].

2. *Massive scalability* A multi-user cloud application may potentially provide service to many thousands or even millions of mobile users, and all sessions must be individually protected. In an extreme example, the social network Facebook, which has characteristics of a private cloud, has over 500 million active users, 40% of whom connect from mobile devices, and half of whom log into the system in any given day and spend more than 55 minutes on the site, on average [5]. The scalability and efficiency of key distribution is critical to the feasibility of a secure cloud system of such scale. Although a cloud may theoretically have the computational resources to scale its cryptographic functions, such task allocation overhead has finite limits; it is at the expense of the provider's clients that require resources to run their own business logic.

3. *Fine-grained access control* Members of the same group, such as a work project team or a social community of common interests, will typically require access to a common data partition resident in the cloud storage system. It is desirable for data permanently stored inside the cloud to be segmented into data partitions; the objective is to subdivide and isolate all cloud data and enforce appropriate access rights on each portion. Each data partition may be separately protected by its own partition-specific group key; or, an authorized user in the group may protect records within the partition with the user's own secret key. A single user may also be authorized to access multiple data partitions within the cloud. Hence, it is insufficient for a user to be provisioned with a single password to control all of the user's data; it may be required at the data partition or even record level. In a large-scale enterprise system, users may be created or removed in significant numbers daily, as permissions are naturally granted and revoked.

2.1 Threat model

It is important to consider the unique security risks present in a cloud environment in order to be able to find adequate security solutions. Clients need assurance of the existence of sufficiently robust security and privacy features in a cloud system before committing to it tasks that add core value to an organization and cannot be placed at risk. Numerous threat scenarios have been identified [6, 2], including: the possible unauthorized access of confidential client data stored in the cloud; the lack of privacy of deployed application logic, either in static or runtime form; and the interference or side-channel attack of the customer's application by another party's malicious application running concurrently on the same cloud server.

Communication security for cloud systems, of particular interest in this work, ensures that unauthorized persons cannot read or manipulate data that is in transit to or stored in the cloud. A related concern is that data may be automatically replicated for reliability or retrieval performance reasons, and remain in storage in the cloud indefinitely, thus requiring strong encryption not only when it is *in-flight* but also when it is *at-rest*. Additionally, user identity management and authentication are considered crucial to realizing safe computing in the cloud. 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.

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 material, 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. The focus here is on communication security rather than the integrity of stored cloud data, which may require the addition of digital signature record-keeping. *Forward secrecy* is always desirable, in that a user whose access is revoked will be unable to access information encrypted using the key material still in the user's possession. For some applications, *backward secrecy* may also prove useful, where a party joining the authorized user set does not immediately obtain access to resources that were encrypted for the benefit of the preceding membership; it may be useful in limiting legal liability for known information, for instance.

2.2 Resource constraints

The challenge with mobile devices is that they possess scarce resources such as limited battery life that is shortened by wireless data transfers and long processing tasks. The Apple iPhone 3GS, for example, includes an internal battery with a capacity of only 1219 mAh. This requires the minimization of the amount of communication exchange in key distribution, usage, and redistribution based on membership change. Uploads conducted on 3G wireless networks consume considerable energy due to a typical wireless radio remaining in a high-power active state after transfer (to maximize responsiveness and minimize the signalling costs of additional transmissions) [7, 8]. In addition, the CPU of a smartphone is considerably slower than that of a server; the same iPhone possesses an ARM processor clocked at only 600 MHz. The system model is thus asymmetric; the cloud server has much greater computational ability than a mobile client to process a security protocol. 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 [9]. The amount of computation associated with cryptographic functions must be minimized to improve responsiveness of the user interface and to limit battery drain.

Despite the asymmetry, cryptographic operations will incur a significant computational penalty on a server that had no security to begin with. In one test, Microsoft measured the performance implications of client authentication on ProLiant web servers [10]. The throughput was approximately 10 times worse when using basic SSL authentication than anonymous requests without authentication; the response time was approximately 4 times worse. Any cryptographic protocol must therefore be sufficiently lightweight in nature, and key updates must be relatively small and infrequent; otherwise, a cloud provider may be ill-equipped to handle this traffic. In February 2008, Amazon's S3 service experienced an outage that affected the availability of thousands of popular sites, including that of Twitter, due to the provider's inability to handle an unexpected surge in processing authentication requests; these cryptographic operations were found to consume more resources than all other request types [11]. The advantage of a cloud system is that, if designed properly, it can take advantage of its inherent scaling property to carry out cryptographic work; for instance, thousands of Amazon EC2 instances can be commissioned within a matter of minutes. A mobile device user does not enjoy the same capability.

3 Related work

There is a clear need for a scalable and efficient key management solution for cloud computing systems, but so far it has not been fully addressed in commercial cloud systems. The Organization for the Advancement of Structured Information Standards (OASIS) has proposed the Key Management Interoperability Protocol (KMIP) for unified cloud management [12]. It addresses the issue of interoperability of key management services, and defines a single low-level standardization protocol for communication between enterprise key management systems and enterprise applications; however, it fails to account for the unique scalability potential in cloud systems and the performance problems that can result. There is agreement over the need for a scalable key management model applicable to today's cloud systems [13].

Other aspects of communication security are already addressed, but in a fragmented and insufficient manner. OpenID is an open-source *Single Sign-On* (SSO) solution that permits a single login to access different sites and resources. A client can choose from any trusted OpenID provider, and has the freedom to change the provider at any time. A fundamental and limiting characteristic is that effectively the same password is used to access multiple sites, with no fine-grained access control. With user federation, authentication in clouds may be accomplished through the use of the Security Assertion Markup Language (SAML), permitting each organization to manage authentication for its own users, as well as between other sites using trust relationships. Although it has SSO capability, it does not address scalable key management within a single cloud provider.

A traditional approach to communication security has been centralized key management, which requires public-key certificates to be generated by the authority and deployed to all users before communication can occur. In a highly scalable system, the authorization server may become overloaded as a result of this responsibility. Security enforcement based on monitoring of user behaviour can mitigate these performance concerns, at the cost of some amount of security. For instance, in *TrustCube*, a star-shaped, or centralized, authentication system provides implicit authentication by monitoring user behaviour, and falls back to another authentication method such as OpenID if a user violates policy norms [14]. The persistent problem with these centralized solutions is that the cloud provider must be entrusted with all aspects of this system, including aggregated data on user contexts and activities, thus relaxing the trust model to a great extent. In Certificateless Public Key Cryptography (CLPKC) [15], the Key Generation Centre (KGC) residing in the cloud does not have access to users' private keys, but the KGC would need to ensure that the partial private keys would be delivered securely to the right users using some secure, or out-of-band, transport.

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, it may be broadcasted in encrypted form to all interested users, with each message being encrypted with a different key corresponding to the recipient. If the 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.

Even if a centralized scheme based on some form of group keying is hypothetically found to offer efficient key delivery, and provides for trusted key escrow, the high turnover of cloud user membership will likely pose a great challenge; expensive re-keying operations are normally required whenever group membership changes, and all users are necessarily involved in the operation. For instance, a stateful scheme such as Logical Key Hierarchy [16] 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. The processing time per request scales linearly with the logarithm of group size, and the signing of rekey messages increases server processing time by an

order of magnitude, which is significant overhead.

Another approach is distributed key management, which may be effectively accomplished using the concept of *partial keys*. Based on an identity-based encryption scheme, it was suggested that distributed public key generators (PKG's) could function as decryption servers in the context of threshold decryption [17]; essentially, each PKG would hold a share of a master key. This would require each PKG to be involved in communication security at all times, because the key share would need to be re-distributed for each new ciphertext; this is inefficient in a cloud scenario. It has also 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 PKG's [18]. 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 member is still responsible for assembling a key from multiple decryption servers and distributing it, which entails expensive communication. Another issue is that of key revocation. A mediator is required to complete the key for each decryption; this would present a bottleneck in a cloud system.

As demonstrated in this work, data re-encryption is a viable mechanism for controlling access to data stored in the cloud. Re-encryption has previously been applied to an encrypted file storage system, where a content owner encrypts blocks of content with unique, symmetric content keys that are then encrypted using an asymmetric master key to form a "lockbox" [19]. Users download the encrypted content from the block store, then communicate with an access control server that decrypts the lockboxes protecting the content. Critically, the content owner decides which users should have access to the content and gives the appropriate delegation rights to the access control server, which could presumably be located in the cloud. 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. One concern with this approach is that the content owner manages access control for all other users, which is a great burden on communication 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 one of the novel models to be presented in this work, 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.

A related work proposes the merging of attribute-based encryption 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 [20]. This scheme has important differences to the cloud-based re-encryption scheme that will be proposed, however; these differences prove disadvantageous in a mobile-based environment. The data owner (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 be more efficient. 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. Finally, there is 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. 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 [21]; 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 [22]. 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 solution in the proposed work avoids the inefficiency of peer-to-peer key distribution in this manner.

Encrypted file storage systems exist, such as SiRiUS [23] and Plutus [24], but their designs are rooted in a traditional client-server setting. They typically offload all cryptographic operations, including key generation, to clients, rather than the server; this characteristic is the opposite of what is desired in a cloud-based system accessed by resource-constrained mobile users. Constant availability of data owners is required to re-generate and distribute encryption and signature keys, but mobile users suffer from transient connectivity and thus this cannot be assured.

4 Scalable key management

The overall goal of this work 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. Three leading key management models are presented with differing design philosophies reflected in the choices; the underlying techniques are based on existing proposals but have been adapted here to a cloud computing system, with appropriate modification, with actors and roles assigned as appropriate, and new improvements proposed as noted. In the first scheme, encryption keys are stored by the cloud provider, the fundamental and relaxed assumption being that it is trusted. In the second scheme, this assumption does not hold, and the keys are managed by a small set of authority entities belonging to the client. In the third scheme, a trusted manager re-encrypts data for recipients as a means of controlling access. The analysis of these schemes will aid in determining a direction for improved security methods. Table 1 summarizes the notation used throughout.

4.1 Centralized key management

In the first model, a central authority manages a single key store and acts as a central hub of communication with users that require keys. This central authority may be deployed as an authentication centre inside the cloud. To simplify the general problem of certificate management, Identity-Based Encryption (IBE) was invented [17]. The public key used in this scheme is an index value that can uniquely identify a shared data segment. Querying a certificate authority for it (as with RSA) is not required, thus reducing communication. This centralized scheme is illustrated in Figure 1. The central authority for key management functions is the cloud provider, which manages an *access control list* (ACL) specifying which users have access to the data stored inside the cloud. A user must request permission from the cloud provider to access any data resources, and the cloud provider fulfills the role of the authenticator. Data that is permanently stored in the cloud may be optionally encrypted at rest, and the user is responsible for obtaining the key to

Table 1: Legend for notation used in algorithmic descriptions of models.

Symbol	Description
P	Cloud data partition
ID_P	Public identifier of data partition P
U_P	User group with authorized access to P
U_{P_X}	Sub group of U_P containing user X in hierarchical access control
M	Manager or trusted proxy
A, B, C	Users Alice, Bob, Charlie
m	Plaintext
C_X	Ciphertext encrypted using key X
σ_X	Signature of user X
$h(x, y)$	Secure one-way hash function with input parameters x, y
r	Random integer
PK_M	Public master key belonging to the cloud provider
SK_M	Private master key belonging to the cloud provider
PK_X	Public key of entity X
SK_X	Private key of entity X
PSK_X	Partial private key of entity X
$\hat{SK}_{P_A}$	First-level (incomplete) secret key of subgroup U_{P_A}
SK_{P_A}	Second-level (complete) secret key of subgroup U_{P_A}
$S(K)$	Binary length of key material K

unlock it. Irrespectively, data is mandatorily encrypted while in transit from and to the cloud over the unsecured network. Furthermore, the data residing on the server is divided into data partitions that are each encrypted with a different key. Multiple users may share the same key to gain access to the same data partition. A controller that runs as a cloud application services all user requests, and utilizes the ACL to grant access to individual data partitions. The controller reads from a key database that is stored directly within the cloud. For instance, consider the IBE scheme based on BDH (Bilinear Diffie-Hellman) [17]. User Alice, designated as A, has been pre-authenticated by the cloud provider. A wishes to share encrypted data that is permanently stored inside the cloud, within a data partition P, with other members of the authorized user set U_{P_1}, including Bob, or B.

4.1.1 Key generation

The cloud provider, as a trusted third-party, assumes the role of a public key generator (PKG), which specifies groups $\mathbb{G}$ and $\mathbb{F}$ of prime order q generated by $g \in \mathbb{G}^*$, the bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{F}$, and ideal random hash functions H_1 and H_2:

$$H_1 : 0, 1^* \to \mathbb{G}^*, \quad H_2 : \mathbb{F} \to 0, 1^l$$

where l is the length of the plaintext. Recall the *bilinearity property*, where $\forall P, Q \in \mathbb{G}$, and $\forall a, b \in \mathbb{Z}_q^*$, $e(aP, bQ) = e(P, Q)^{ab}$. The provider chooses a random asymmetric master secret key $SK_M \in \mathbb{Z}_q^*$, belonging to the provider itself and always kept secret, and then calculates the

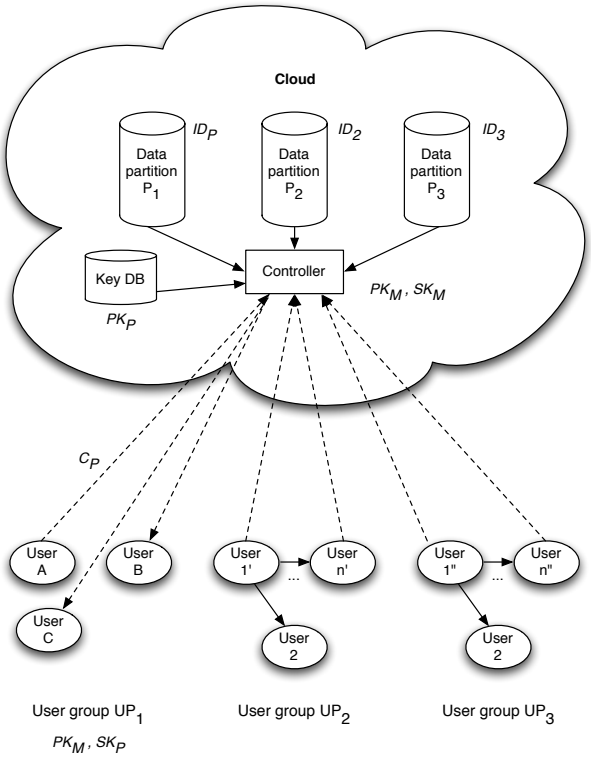

Figure 1: Centralized key management model.

corresponding public master key PK_M, which is shared with all users:

$$PK_M = SK_M \cdot g$$

The PKG is now ready to distribute PK_M, as well as key-related parameters, to the authorized user set. The set of parameters includes descriptions of $\mathbb{G}, \mathbb{F}, H_1, H_2$, and PK_M (such as the binary length $S(K)$ of the key material).

4.1.2 Encryption

In the *set-up* phase, A, the requestor, obtains PK_M and key parameters from the cloud provider's key centre; this is a one-time task. For this particular data encryption operation, A also requests, directly from the cloud, the global identifier ID_P corresponding to the data partition intended to be shared. A proceeds to encrypt a message $m \in \{0,1\}^l$ as C_P using PK_M in combination with ID_P. A chooses a random value $r \in \mathbb{Z}_q^*$ in the encryption.

$$\begin{aligned} U &= rg \\ V &= H_2(\hat{e}(Q_{ID}, PK_M)^r) \oplus m \\ \text{where } Q_{ID} &= H_1(ID_P) \end{aligned}$$

A uploads ciphertext $C_P = (U, V)$ to the cloud provider. No additional communication with B or the cloud is necessary at this point to carry out the encryption. The encrypted message C_P can be uploaded and stored securely in the cloud.

4.1.3 Decryption

The ultimate recipient of the message, user B, authenticates with the cloud provider and requests the private key SK_P from the cloud's key centre. This preliminary key setup activity must be done only once for each data partition to which B requires read and write access.

$$SK_P = SK_M \cdot Q_{ID}$$
$$\text{where } Q_{ID} = H_1(ID_P)$$

B then downloads the message from the data partition in the cloud and decrypts it using SK_P.

$$M = V \oplus H_2(\hat{e}(SK_P, U))$$

4.1.4 Analysis

The application of IBE presented here varies from the original concept of point-to-point communication. The PKG generates a secret key for each data partition, rather than each user, so that any number of users may share the same key and access a common partition; this is in contrast to a single originator and consumer sending data in one-way fashion. The cloud provider is entrusted with *key escrow*: it manages the secure storage and distribution of the master public and private key pair. During the initial provisioning, the private keys may be distributed to all eligible partition users based on an ACL as each user performs authorization and initiates a key request; SSL may be used for this purpose.

This scheme is relatively straightforward in that key management is managed as a cloud service, and all communication occurs directly between the user and cloud provider. It does not require any additional network components or roles to exist within the system. An advantage of IBE is that only the cloud provider generates a key for each data partition, and that key can be locally derived on each device with a commonly understood index. An expensive pre-distribution of authenticated keys is unnecessary, unlike with a public-key infrastructure utilizing RSA. Additionally, the cloud provider does not need to store a list correlating key pairs to data partitions. However, this scheme suffers from a major deficit: the cloud provider must be trusted entirely. Because the cloud's key centre generates all private keys, it has the ability to decrypt all data and communication in its domain. In addition, this model assumes that the cloud provider can be trusted to permit access only to authorized users so that they may retrieve their private keys; this wide-ranging premise of trust is not widely accepted in the IT circle, and is only appropriate for less sensitive data. Even if the cloud is so trusted, the cloud-based authenticator remains a critical point of failure if attacked. If its keys are stolen, then all communication is compromised. If it suffers from a denial-of-service attack, it cannot distribute private keys. It is also subject to discovery through law enforcement means or a court order, whether agreed to or even known to the client or not, leading to the possibility of abuse. A similar problem occurs when a third-party service is responsible for key escrow, such as a contracted security provider. Even if the key storage is external to the cloud, it represents a single point of vulnerability. Communication and authentication functions could be disrupted through power failure or a denial-of-service attack.

The cloud is responsible for key generation, authentication, and key requests. If a user's private key is compromised or the user's access to the cloud is revoked, the centralized PKG must re-generate and re-distribute new encryption keys to all valid users in the domain. It must have sufficient computational resources and bandwidth capacity to handle these tasks for its entire user population. Although clouds are designed to scale accordingly, authentication is a considerably more computationally intensive task than hosting typical web server functions. Bandwidth capacity

is also an important consideration. The primary bandwidth cost in the centralized scenario occurs with respect to key generation due to changes in membership. It is desirable for a new user to be granted permission to access an existing secure resource. Likewise, if a user leaves the membership of a group, then that user's access rights must be revoked. Forward and backward secrecy must somehow be preserved. Fundamentally, if public key encryption is employed, then the public master key must be re-generated by the cloud provider and re-distributed to all users, at great expense.

Note that the public identifiers of all data partitions must be computable, in order to utilize IBE. In addition, if the same data is to be decrypted by multiple users, then they must each be able to obtain the same private key based on the same public identifier. If it is not possible to readily identify the user or data partition to be used as the basis for generating an encryption key, then a fallback to a more traditional public key cryptography scheme such as RSA would be required; the public keys of participants would be provisioned by the cloud provider acting as a key authority, further adding to the communication cost.

Overall, the first problem mentioned is the most vital one: most IT organizations would be unwilling to implicitly trust the cloud provider with key escrow and hence unrestricted access to all data. Even if the cloud provider was to be considered trustworthy beyond doubt, the keys could be stored in a disadvantageous jurisdiction or potentially be subject to attack by an insidious employee or malicious other client executing applications side-by-side in the cloud.

4.2 Multi-level key management

In order to reduce the authentication traffic load on the cloud server, the key store may be relocated from the cloud provider's domain to the client's. To this end, a multi-level key distribution mechanism may be utilized. The entire user population no longer relies on obtaining keys directly from the cloud provider. Users are segmented into populations called *groups*, each of which has read and write access to a different data partition. Data partitions within the cloud are encrypted so as not to reveal information to the cloud provider. A trusted intermediary called a *manager* or *authority* is responsible for user authentication within each group; the manager, rather than the cloud provider, is entrusted with key escrow. In terms of the network model, the manager is situated outside the domain of the cloud provider, but runs on a secure server. A single manager is responsible for a group of users of any size, and is responsible for all aspects of its security: it generates and holds all private keys, allowing read and write access to the data partition; it uses the ACL to verify group membership; and, it is responsible for authenticating the users belonging to its group, thus allowing them entry. A diagram of the multi-level key management scheme is provided in Figure 2.

Managers are disassociated from the cloud; they each oversee an assigned user group with access rights to a specific data partition in the cloud. Once users discover their secret keys, they communicate directly with the cloud controller to carry out operations. A scheme called Dual-Level Key Management (DLKM) is useful as the basis for this model [25] [26]. It was originally intended for grid computing, a pre-cursor to cloud computing. The scheme has been adapted here for cloud use so that key parameters, used to generate secret keys, are stored in a directory in the cloud for readier dissemination. Also, versioning ensures that users are able to download the latest parameters and re-generate their keys only when required.

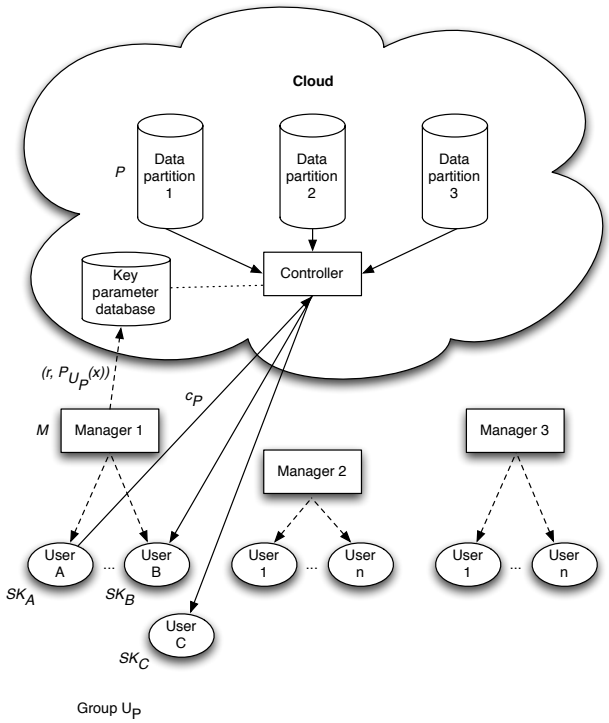

Figure 2: Multi-level key management model.

4.2.1 Key generation

Suppose that a group of users U_P are managed by a manager M. The manager constructs an ACP (Access Control Polynomial) $A(x)$ in finite field $F_p[x]$:

$$A(x) = \prod_{i \in U_P} (x - h(SK_i, r))$$

where SK_i are the secret keys assigned to each member in U_P, $h(x, y)$ is a public one-way secure hash function that is a random oracle, such as SHA-2, and r is a random integer from F_p. Note that when x is substituted with $h(SK_i, r)$ from a valid user possessing SK_i in U_P, then $A(x)$ is 0; otherwise, it is a polynomial that evaluates to a random and useless value. During the set-up phase, the manager must distribute secret keys to all current authorized users; namely, SK_i for $i \in U_p$. The manager then selects a random group key SK_P for U_P, and computes polynomial $P(x)$:

$$P(x) = A(x) + SK_P$$

Suppose that in group U_P, there exist users Alice and Bob, or A and B, holding secret keys SK_A and SK_B. A and B are both managed by the same manager. The polynomial will thus equate to:

$$P_{U_P}(x) = (x - h(SK_A, r))(x - h(SK_B, r)) + SK_P$$

The manager uploads the tuple $(r, P_{U_P}(x))$, representing the key parameters, to the cloud provider; the provider assigns it to data partition P, and stores the tuple in a public directory accessible to all users. This key database can also be stored in a private cloud, behind the organization's firewall,

or on a server that is considered trusted. Knowledge of these key parameters alone is insufficient to derive keys for decryption. The manager also distributes the secret keys SK_i to all users in U_P, to allow access to the data partition. That is, upon user authentication, the manager will issue SK_A to A and SK_B to B. For this to occur, each user must first authenticate with the manager, which confirms the user's identity based on the ACL. In contrast to the previous centralized key scenario, where the cloud provider maintains all keys and authorizes access to data stored in the cloud, the manager now fulfills this function.

4.2.2 Encryption

User A accesses the data partition P in the cloud and reads the tuple stored for it. A finds that access to P is governed by the random access value r, so she generates the partition access key by computing the hash of her own secret key SK_A and substituting it into the public polynomial:

$$SK_P = P_{U_P}(h(SK_A, r))$$

A encrypts message M as ciphertext C_P using the symmetric partition group key SK_P. She then authenticates with the cloud and stores C_P in partition P.

4.2.3 Decryption

B derives the partition key using the same approach as A:

$$SK_P = P_{U_P}(h(SK_B, r))$$

As B now possesses the key to unlock cloud data, he can issue a request for it. B authenticates with the cloud, downloads C_P, and then decrypts it using SK_P.

4.2.4 Key re-generation

Suppose that a third user, Charlie, or C, joins group U_P. After allowing this change in membership, the manager creates a new, modified ACP $A'(x)$:

$$A'(x) = A(x) \cdot (x - h(SK_C, r))$$
$$P'_{U_P}(x) = A'(x) + SK_P$$

The new tuple $(r, P'_{U_P}(x))$, updated to include newcomer C, is sent to the cloud provider to replace its current one. Note that the existing users A and B need not obtain a copy of the latest tuple, as they are already in possession of SK_P. Whenever a new user is added, this partition group key does not change. However, when a user's access is revoked, then a new group key must be generated. Suppose that C leaves the group U_P. The manager selects a new random group key SK'_P and a new random r' value and recomputes the polynomial as follows:

$$P''_{U_P}(x) = A'(x) + SK'_P$$

The manager then multicasts $(r', P''_{U_P}(x))$ or stores the tuple in the cloud, as before. C, who has left, will be unable to extract SK'_P from $P'(x)$. The manager has the final determination as to which users have access to the partition.

Because the group key has been modified upon access revocation, from SK_P to SK'_P, the existing data stored in the cloud partition and encrypted with the first iteration of the group key will only be accessible by users who retain that key. For all current authorized users to gain access to data encrypted using the new key SK'_P, the cloud data would need to undergo re-encryption, which is not addressed in the DLKM protocol. Ideally, the data should not be decrypted at any stage to permit the provider access to it.

4.2.5 Analysis

The primary advantage of the multi-level approach is that trusted managers that are outside of the cloud domain manage the keys, and not the cloud provider. Additionally, each manager handles the authentication of only a limited set of users that interact with its own data partition. Thus, this system greatly reduces the amount of communication with the cloud required for the authentication function, and the resultant expense for the client. The manager may be realized as an authenticator behind a company's firewall, in a private cloud environment, or as a trusted off-site party such as a contracted encryption services firm. Another benefit is that the usage of symmetric keys allows users to more quickly perform encryption operations on cloud data.

The protocol efficiently handles changes in user membership. Key re-generation occurs only when a user's access rights are revoked, and when a breach of trust is found to occur; for instance, if a user shares a secret key with another user. It is not necessary for the manager to incur the communication cost of redistributing the key parameters in point-to-point fashion with all users; this can be accomplished by updating the key parameters stored in the cloud, as described. Also, it should be observed that a group key must be generated for a single partition and its associated user membership only; a system-wide update does not occur.

In addition to the described protocol, it may still be desirable for the cloud to perform authentication of each user prior to granting read or write access, irrespective of the validity of the partition key held by the user. For instance, public key cryptography can be used to create certificates. The user generates a public and private key pair, and then creates a CSR (Certificate Signing Request) that is then sent to the authority. The CSR consists of the user's identity and public key, and is signed by the private key. The manager functions as a classic trusted third-party certificate authority in this case. The manager verifies the integrity of the CSR, and generates a signed certificate for the user. Using this token, the user can then proceed to contact the cloud provider to perform normal operations such as read and write requests on encrypted data. To do so, each user receives a unique certificate token. The use of certificates is optional, however.

5 Manager-based re-encryption

A key management scheme is now described that is closely based on the original work suggested in [19]; however, it has been mapped to a cloud computing system. 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 6. Some novel variations of the original scheme are still suggested here, however. The scheme permits access to a common data partition in the cloud among multiple users, ensures confidential data storage not privy even to the cloud provider, 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.

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 user group can decode it using the reader's own private key. Table 2 summarizes the notation used throughout.

Table 2: Legend for notation used in algorithmic descriptions of models.

Symbol	Description
P	Cloud data partition
U_P	User group with authorized access to P
M	Manager or trusted proxy
A, B, C	Users Alice, Bob, Charlie
m	Plaintext
C_x	Ciphertext encrypted using key x
PK_{X_v}	Public key of entity X (with version v optionally specified)
SK_{X_v}	Private 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

5.1 System operation

5.1.1 Key generation and encryption

Consider a proxy re-encryption scheme [19], based on the BBS encryption method [27] and the El Gamal crypto-system [28]. The proof of the underlying encryption technique is presented in [19], and is assumed here. As shown in Figure 3, the manager generates public and private 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 key SK_X is randomly selected for each user $X \in U_P$. 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 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, Alice, encrypts a message m using the public key PK_P of the data partition where it is to be stored, and uploads the cipher-text C_p to the cloud, so that it is stored in encrypted form in partition P. The cloud provider will be unable to extract the original content m.

$$m \in \mathbb{G}_2, \text{ random } r \in \mathbb{Z}_q^*$$
$$C_p = (Z^r \cdot m, g^{pr})$$

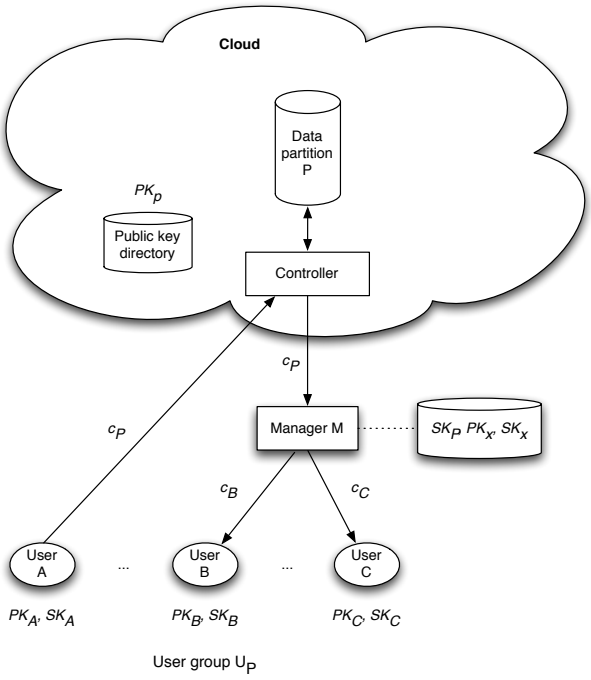

Figure 3: Model of key management using manager-based re-encryption.

5.1.2 Re-encryption

Suppose that a user B, 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 Access Control List (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 C_p from the cloud, and computes a re-encryption key using B's private key SK_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:

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

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 C_p as C_b and sends it to B directly.

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

5.1.3 Decryption

The recipient B can then decode the ciphertext C_b using his own private 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 C_a using her private 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 [19], where the content published by 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 C_p to C_c is required.

5.1.4 Data flow

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

$$\boxed{A} \underset{\rightarrow}{C_p} \boxed{P} \underset{\rightarrow}{C_p} \boxed{M} \underset{\rightarrow}{C_b} \boxed{B}$$

The cryptographic operations explained in this section are shown visually in Table 3.

Table 3: Summary of operations in manager-based re-encryption.

	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 $C_p = (Z^r \cdot m, g^{pr})$, and sends it to the cloud.			
3		Stores C_p, and sends a copy of it to M on request.		
4			Computes $RK_{P \to B} = g^{\frac{b}{p}}$. Re-encrypts C_p as $C_b = (Z^r \cdot m, Z^{br})$.	
5				Downloads C_b and decodes $m = \frac{Z^r \cdot m}{(Z^{br})^{\frac{1}{b}}}$ using SK_B.

5.1.5 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 C_c. If Charlie leaves the group, then the manager removes him from its access list; it will no longer re-encrypt data for C on a retrieval attempt.

5.2 Analysis

A chief advantage of this model lies in 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 cloud 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*.

A disadvantage of this approach 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 [29]. 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 for this purpose will be described later.

It is useful to observe that a significant advantage of re-encryption over public-key encryption is that for extremely sensitive data, if Alice were to encrypt it using her own locally-generated key rather than the public data partition key, and upload it to the cloud, she could then have the manager re-encrypt the data simply by supplying appropriate re-encryption keys for the intended recipients. The manager would not need to know Alice's own encryption key, and Alice would not need to incur the cost of the re-encryption herself.

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 in place; 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.

Because the manager stores all decryption keys, it must be fully trusted; hence, it is a point of vulnerability. However, a private cloud would typically possess the same safeguards as that of a public cloud. Collusion between the manager and users is not deemed to be a concern, as the manager and all its authorized users are expected to belong to the same client organization and share equal access to the data partition.

5.3 Possible and novel variants

5.3.1 User-managed keys

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 C_a to the cloud. Upon data retrieval, the manager would be required to decrypt C_a using its own retained copy of SK_A, then perform the re-encryption for another user, such as B, using $RK_{A \to B}$:

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

This technique would allow Alice to retrieve data directly from the cloud that she could then decrypt without the aid of the manager. This 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, as described earlier. Even so, no key re-distribution would need to occur.

5.3.2 Partial ciphertext fetch by user

If the manager introduces too much latency in 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. The manager's critical role in the described protocol is to perform the re-encryption task. However, the $Z^r \cdot m$ subcomponent of the encrypted ciphertext C_p stored in the cloud is not directly involved in this operation; it may be downloaded from the cloud by the recipient B directly. B will then await the second component Z^{br} from M. In this way, M avoids the overhead of fetching C_p in its entirety from the cloud. An undesirable side effect is that if B leaves the group, he can continue to download and access encrypted data in the cloud. This is solved in the next model, where the cloud data undergoes a transformation preventing this.

6 Cloud-based re-encryption

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 encryption operations are handled by the cloud provider, which is highly scalable.

6.1 System operation

6.1.1 Key generation and encryption

Refer to Figure 4. 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 as described in Section 5, using the BBS scheme; 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, Alice, or user A, wishes to store encrypted data in the cloud. A encrypts a message m with PK_{P_0}. A then uploads the ciphertext C_{p_0}, and any optionally associated policy settings, to the cloud provider: $C_{p_0} = (Z^r \cdot m, g^{p_0 r})$. The data is stored in P, in encrypted form.

6.1.2 Decryption

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

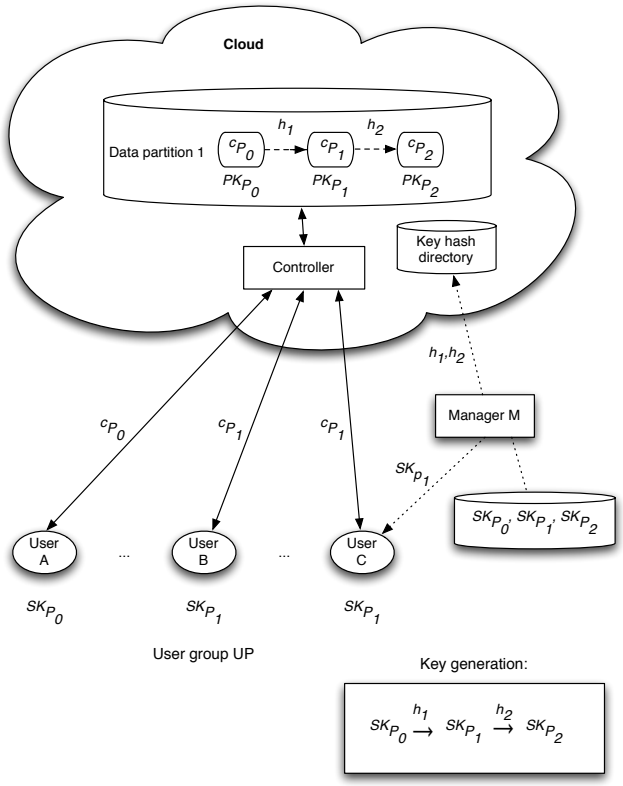

Figure 4: Model of key management using cloud-based re-encryption.

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 wireless channel. All users in U_P may retrieve the message uploaded by A to the cloud, by directly obtaining C_{p_0} from the cloud provider, and using the same shared decryption key SK_{P_0}. Thus, in this model, *second-level encryption* is demonstrated [19]; 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 cloud provider, 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.

6.1.3 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 as 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 C_{P_0} 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 C_{P_1}, 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 C_{P_0} is transformed into C_{P_1}:

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

C can now proceed to download and decrypt the message as: $m = \frac{Z^r \cdot m}{(Z^{p_1 r})^{\frac{1}{p_1}}}$.

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 possession. If not, when the client requests the ciphertext from the cloud provider, he can request that it be re-encrypted to the same version of the key that is in the user's possession, assuming that the key corresponding to the ciphertext is more recent. If it is not, 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.

6.1.4 Data flow

To summarize, the flow of ciphertext in the system between two users is as follows, with the manager not playing the role of an intermediary any more in the communication:

$$\boxed{A} \underset{\rightarrow}{C_p} \boxed{P} \underset{\rightarrow}{C_p} \boxed{B}$$

The cryptographic operations described in this section are summarized in Table 4.

6.2 Analysis

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 previous scheme, 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 with 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 decreases). 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. As user memberships typically increase, not decrease, this will be the normal course of events.

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

Table 4: Summary of operations in cloud-based re-encryption.

	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 $C_{p_0} = (Z^r \cdot m, g^{p_0 r})$, and sends it to the cloud.				
2		Stores C_{p_0} and its associated version 0.			
3				Downloads C_{p_0}. 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.		
5		Re-encrypts C_{p_0} as $C_{p_1} = (Z^r \cdot m, Z^{p_1 r})$ using $RK_{P_0 \to P_1}$, and updates version to 1.			
6					Downloads C_{p_1} and decodes $m = \frac{Z^r \cdot m}{(Z^{p_1 r})^{\frac{1}{p_1}}}$ using SK_{P_1}.

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 hash values to arrive at the latest key, and discard all remnants of its history. Thus, only one secret key must be locally stored for each data partition that the user interacts with.

Note that the original re-encryption protocol based on BBS [27] 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. In other words, 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 only retain 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, $C_{P_{x+1}}$ would be re-encrypted directly to $C_{P_{x+2}}$, rather than from the original C_{P_x}. However, 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 [19]; 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 C_{p_0} in the cloud for use in all future re-encryptions, and incur a storage cost. The provider may still cache the ciphertext resulting from the most recent re-encryption, however, for immediate access to it.

The main drawback with this approach is the re-encryption task required whenever group membership changes, which may be 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. However, there still exists the risk of the key being illegitimately shared by a misbehaving (yet authorized) user with that of an unauthorized user. 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.

6.3 Possible variants

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, 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. 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. 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 transfer, as only a small one-time transfer of key material is needed.

7 Evaluation and implementation of models

7.1 Qualitative cost comparison

The processing and storage costs of the transactions in the centralized and multi-level key management models are shown in table 5. The computational complexity of each model is summarized. Note that a bilinear pairing operation (BP) based on a Weil and Tate pairing is several times more costly than a scalar multiplication (M) [29]. In the case of encryption and decryption, assume that the user's existing key is still valid from the previous session. The cost of key generation and re-generation, due to user revocation, is reflected from the viewpoint of the server or key generation facility (either in the cloud provider or an external manager node), as well as the user.

The scalability of the these two mechanisms is limited due to the key re-generation and re-distribution that must occur for the entire user population whenever new users join or existing ones leave. The centralized scheme requires full trust in the cloud provider to securely manage keys for all users, which is unrealistic. In terms of the cost of computation, the multi-level scheme generates symmetric keys that are less costly to use than the asymmetric keys in the centralized scheme. The key generation entails more operations in the multi-level scheme to re-generate the Access Control Polynomial, but only on user revocation, not on the addition of a user to the membership.

In contrast, the processing and storage costs of the transactions in the two proposed re-encryption models are shown in Table 6. 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.

7.2 Implementation results

In order to understand the execution cost of the protocol on real hardware, the cloud-based re-encryption algorithm described here was implemented in Java using jPBC (Java Pairing-Based Cryptography Library) [30], a porting of the PBC (Pairing-Based Cryptography Library) in C [31]. The encryption, re-encryption, and de-cryption tasks, as described earlier, were timed on different platforms; portability was provided by Java. 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 10.6.7. The smartphone platform consisted of a Google Nexus One phone with a single-core 1 GHz Qualcomm QSD 8250 Snapdragon ARM processor and 512 MB of memory, running Android OS 2.3.4. 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

Table 5: Storage and communication costs of centralized and multi-level communication models. All cryptographic operations are over multiplicative group $\mathbb{G}$. The operations include: Hashing (H), Exponentiation (E), Bilinear pairing (BP), and Multiplication (M).

Computational complexity		
Description	Centralized model	Multi-level model
Key generation (server)	M	$H + M$
Key generation (user)	-	$H + M$
Encryption	$BP + E + H$	H
Decryption	$BP + H$	H
Key re-generation (server)	$M \cdot U_p$	$(H + M) \cdot U_p$
Key re-generation (user)	-	$H + M$
Computational costs		
Description	Centralized model	Multi-level model
Key generation	1 for each user in the system (generated by cloud)	1 for each user in subgroup (generated by manager)
Re-encryption	(none)	(none)
Access model		
Description	Centralized model	Multi-level model
Data fetch	Direct-from-cloud	Direct-from-cloud
Storage costs		
Description	Centralized model	Multi-level model
Key storage	1 for each user in the system (stored in cloud)	1 for each user in subgroup (stored in each manager)

Table 6: Storage and communication costs of re-encryption models. The cryptographic operations include: Hashing (H), Exponentiation (E), Bilinear pairing (BP), and Multiplication (M).

Computational complexity		
Description	Proxy re-encryption	Cloud 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 re-encryption	Cloud re-encryption
Key generation	(none)	(none)
Re-encryption	1 per join/leave (operation done by proxy)	1 per join/leave (operation done by cloud)
Access model		
Description	Proxy re-encryption	Cloud re-encryption
Data fetch	Via proxy	Direct-from-cloud
Storage costs		
Description	Proxy re-encryption	Cloud re-encryption
Key storage	All stored in manager	All stored in manager

with 128 MB RAM, at 10 cents per hour; the actual number of CPU cycles used is internal to the App Engine and not exposed. A "Type A" pairing was utilized in the algorithm; for direct comparison, the field size was reduced to 32 bits to avoid stack overflows during curve generation, owing to the limited heap available on the phone. The average timing results are shown in Table 7. The re-encryption was much more feasible on a cloud instance or a fast desktop computer; in the latter case, it was 38 times faster than on the smartphone. Note that each operation was performed on a 48-bit message block. 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 back-end instances with up to 4.8 GHz CPU and 1 GB memory.

Although these benchmark results are limited, to be fair, they were achieved using libraries that are not highly mature and optimized for a mobile platform; increasing workload sizes to expand the tests incurred the risk of out-of-memory conditions. However, they serve to validate the comparative strengths of mobile devices and clouds that the re-encryption model leverages.

Table 7: Performance results obtained from the implementation.

Cryptographic task on 48-bit data block	Desktop (iMac)	Smartphone (Android)	Cloud (GAE)
Encryption time (ms), using P_0 key	8.3	200.1	8.8
Decryption time (ms), using P_0 key, i.e. before re-encryption occurs	4.9	128.9	2.9
Re-encryption time (ms), from P_0 to P_1 keys	4.2	159.6	3.0
Decryption time (ms), using P_1 key, i.e. after re-encryption occurs	0.5	15.2	0.5

8 Summary

Various cryptographic protocols have been adapted to a cloud computing system model in order to gauge their viability in improving communication security. Appropriate modifications have been proposed to support both public and private clouds, and standard 3G 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. Compared to a fully centralized model based on straightforward RSA, the first identity-based scheme reduces the cost of initial key distribution, but at the expense of requiring complete trust in the cloud provider. The second multi-level key management scheme addresses the trust issue by relocating key storage to a trusted manager entity; however, it fails to solve the problem of re-encryption of data within the cloud to protect it as the user membership changes over time. The third 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 must occur 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, but 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.

Although the focus of this protocol is on data confidentiality, data integrity may be provided through the use of message authentication codes to achieve a holistic security solution. It is still unclear, however, how hierarchical access can be efficiently achieved for users of different access class privileges. Hierarchical access control for a re-encryption-based protocol must support mobile users that require local storage and communication to be kept to a minimum; this is likely a useful future research direction. Also, the question of how to perform useful operations directly on the encrypted data stored within the cloud itself is an open research problem. Fully homomorphic encryption schemes have been proposed but are still largely impractical [32]. However, the proposed work is suitable for the common scenario of a massively scalable application such as a web server delivering encrypted content to mobile users. Finally, it would be interesting to apply the techniques to real-world applications to gauge the performance on varying workloads and user populations.

9 Acknowledgments

This work was supported in part by a National Sciences and Engineering Research Council (NSERC) grant awarded to Dr. Hasan, and an NSERC Alexander Graham Bell Canada Graduate Scholarship (Doctoral) awarded to Piotr Tysowski.

References

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

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

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] 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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inf. Syst. Secur.*, vol. 9, pp. 1–30, Feb. 2006.

[20] 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.

[21] 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.

[22] 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. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.36

[23] 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.

[24] 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.

[25] 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.

[26] X. Zou, Y.-S. Dai, and X. Ran, "Dual-level key management for secure grid communication in dynamic and hierarchical groups," *Future Gener. Comput. Syst.*, vol. 23, pp. 776–786, July 2007.

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

[28] 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.

[29] 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.

[30] [Online]. Available: http://libeccio.dia.unisa.it/projects/jpbc/

[31] [Online]. Available: http://crypto.stanford.edu/pbc/

[32] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.