Re-Encryption-Based Key Management Towards Secure and Scalable Mobile Applications in Clouds

Piotr K. Tysowski
Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
pktysowski@uwaterloo.ca

M. Anwarul Hasan
Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
ahasan@uwaterloo.ca

***Abstract*—Cloud computing confers strong economic advantages, but many clients are reluctant to implicitly trust a third-party cloud provider. To address these security concerns, data may 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 serving the recent trend 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, a model for key distribution based on the principle of dynamic data re-encryption is applied to a cloud computing system in a unique way to address the demands of a mobile device environment, including limitations on client wireless data usage, storage capacity, processing power, and battery life. The proposed cloud-based re-encryption model is secure, efficient, and highly 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 costs 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 model.**

***Keywords*-Distributed systems, mobile computing, security.**

I. 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 by 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 results in equal opportunity of benefits to firms large and small.

Yet, despite all of its economic benefits, the cloud computing model poses very significant risks to its users. 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. 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 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, especially for a mobile user population. 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 a leading key management scheme, that was 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.

The main contribution of this work is a novel solution that 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.

II. System model

The system under study is a *public cloud* provider 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. Each mobile device user typically opens a direct TCP/IP connection to a cloud application portal. 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]. Members of the same group, such as a work project team or a social community having common interests, will typically require access to a common data partition resident in the cloud storage system. The objective is to subdivide and isolate all cloud data into partitions and enforce appropriate access rights on each.

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.

A. 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. 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 important to realizing safe computing in the cloud.

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.

B. Resource constraints

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 must be minimized to maintain battery life and user interface responsiveness; and, the availability of mobile devices may be limited due to connectivity loss. Uploads conducted on 3G wireless networks consume considerable energy due to a typical wireless radio remaining in a high-power active state after transfer [3] [4]. In addition, the CPU of a smartphone is considerably slower than that of a server. The system model is thus asymmetric; the cloud server has much greater computational ability than a mobile client to process a security protocol [5].

Despite the asymmetry, cryptographic operations will incur a significant computational penalty on a server that had no security to begin with. In one test, throughput was degraded by a factor of 10 when SSL authentication was added to web servers [6]. 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. Thousands of instances can be commissioned within a matter of minutes; a mobile device user does not enjoy the same capability.

III. 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 Key Management Interoperability Protocol (KMIP) addresses the issue of interoperability of key management services, but fails to account for the unique scalability potential in cloud systems and the performance problems that can result. OpenID is an open-source *Single Sign-On* (SSO) solution that permits a single login to access different sites and resources, but effectively the same password is used to access multiple sites, with no fine-grained access control.

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, such as in *TrustCube* [7], but the cloud provider must be entrusted with aggregated data on user contexts and activities, thus relaxing the trust model. In Certificateless Public Key Cryptography (CLPKC) [8], 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 partial private keys would be delivered securely to the right users using some secure, or out-of-band, transport. *Broadcast encryption* may be employed, in which the key manager generates symmetric keys for multiple users, but whenever the membership changes, then new keys must be rebroadcast to all users, which is an unrealistic proposition in a highly scalable system.

The high turnover of cloud user membership poses a great challenge; expensive re-keying operations are normally required whenever group membership changes. In the Logical Key Hierarchy [9] scheme, 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. Another approach is distributed key management, where multiple distributed public key generators (PKGs) hold shares of a master key using the concept of *threshold decryption* [10], or portions of a private key are distributed among users [11]. The problem with these approaches is that a user must assemble a key from multiple sources, resulting in expensive communication sessions.

Data re-encryption is gaining traction as 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* [12]. Concerns include the following: the content owner manages access control for all other users, which is a great burden if the owner is a mobile device user; it requires dynamic re-encryption of the same data whenever multiple users want to access it; access rights need not be enforced by individual users; and it is possible for a single user to divulge the keys of all other users to the cloud provider.

A related work proposes the merging of attribute-based encryption with proxy re-encryption while attempting to offload re-encryption activity to the cloud provider [13]. However, the data owner (originator) is involved in generating a key for each new user that joins or leaves the system, which is not only a prohibitive cost for a mobile user, but also impractical due to the user's mobility and hence occasional unavailability. A secret key must be regenerated and re-distributed for each user, in lazy fashion, whenever user revocation occurs. Also, the data re-encryption activity is aggregated in lazy fashion, rather than on-demand. Similar limitations are evident in 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 uses the cloud provider to re-encrypt data on user revocation depending on the attribute keys held by the revoked user [14]; 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. Another method of trusted data sharing over untrusted cloud providers has been proposed that uses a progressive elliptic curve encryption scheme [15]. 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.

Encrypted file storage systems exist, such as SiRiUS [16] and Plutus [17], 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. A more detailed discussion of related work is found in the authors' original technical report [18].

IV. Manager-based re-encryption

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. A key management scheme is now described that is closely based on the original work suggested in [12]; 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 V. 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. Table I summarizes the notation used.

TABLE I

Legend for the symbolic notation used in the description of the key management 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)
$RK_{X \rightarrow Y}$	Re-encryption key for converting from content unlocked by SK_X to that unlocked by SK_Y

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.

A. System operation

1) Key generation and encryption: Consider a proxy re-encryption scheme [12], based on the BBS encryption method [19] and the El Gamal crypto-system [20]. The proof of the underlying encryption technique is presented in [12], and is assumed here. 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. A high-level diagram is shown in Figure 1.

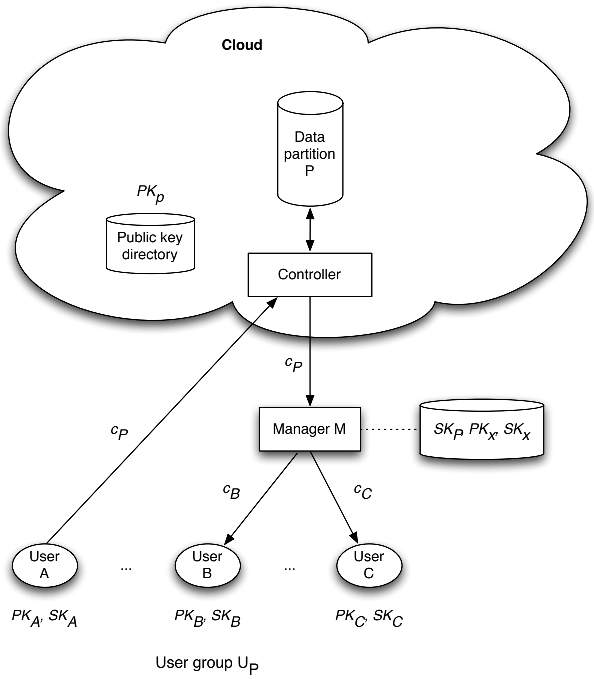

Fig. 1. Model of key management using manager-based re-encryption.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ be groups of prime order q with a bilinear map such that: $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. 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 $m \in \mathbb{G}_2$ using random $r \in \mathbb{Z}_q^*$ and 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, where $C_p = (Z^r \cdot m, g^{pr})$, so that it is stored in encrypted form in partition P. The cloud provider will be unable to extract the original content m.

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 $RK_{P\rightarrow B}$ 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: $RK_{P\rightarrow X} = g^{\frac{SK_X}{SK_P}}$.

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

$$\text{Compute: } e(g^{pr}, RK_{P\rightarrow B}) = e(g^{pr}, g^{\frac{b}{p}}) = Z^{br}$$
$$\text{Publish: } C_b = (Z^r \cdot m, Z^{br})$$

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 $RK_{P\rightarrow A}$ 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 [12], 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 $RK_{P\rightarrow C}$ to carry out a re-encryption of C_p to C_c is required. The flow of ciphertext in the system is as follows:

$$\boxed{A} \xrightarrow{C_p} \boxed{P} \xrightarrow{C_p} \boxed{M} \xrightarrow{C_b} \boxed{B}$$

TABLE II
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.

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

4) 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 C 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.

B. 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, however, is that for each retrieval attempt of a new data block or record, the manager must perform re-encryption using an asymmetric key. A bilinear pairing operation based on a Weil and Tate pairing is several times more costly than a scalar multiplication [21]. 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.

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.

The chief 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.

C. Possible and novel variants

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

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.

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.

V. Cloud-based re-encryption

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. A high-level diagram is shown in Figure 2.

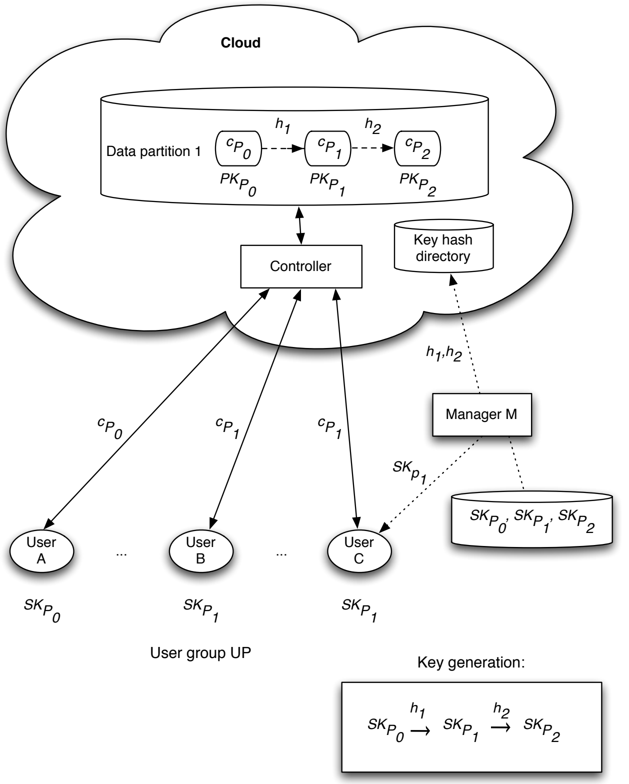

Fig. 2. Model of key management using cloud-based re-encryption.

A. System operation

1) Key generation and encryption: 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 IV, 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.

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 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 [12]; 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.

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. 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}:

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

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 actually in the user's possession, if the ciphertext is encoded with an earlier version. On the other hand, if the ciphertext version is more recent, then the user can re-assemble the correct private key using the hash value chain history H that can be downloaded at any time from the cloud. The user must then perform only a single decryption; multiple decryptions are not required.

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

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

The flow of ciphertext in the system is now as follows:

$$\boxed{A} \;\underset{\to}{C_p}\; \boxed{P} \;\underset{\to}{C_p}\; \boxed{B}$$

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

B. 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 to the cloud while allowing all security requirements to continue to be satisfied. Any authorized user can write and read encrypted data directly to and from the cloud without involvement of the manager or any other proxy, resulting in fast access on a regular basis. Data confidentiality is preserved in this model even when changes to group membership occur. Since a new user is only given the latest iteration of a key and cannot decrypt messages encrypted earlier with older keys, backward secrecy is preserved. 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 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

TABLE III

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}.

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

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

This flexibility would allow the cloud provider to 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}. The cost is that it would allow a newly joined user to collude with the holder of SK_{x+1} and the provider by sharing its private key SK_{x+2}; the cloud provider could deduce $RK'_{PK_{x+1} \to PK_{x+2}}$, as shown, and re-encrypt data for the new user that was not actually intended to be accessed by him. In contrast, the re-encryption protocol based on bilinear maps, as described here, is not transitive, and thus such delegation to new users is not allowed without arbitration from the manager. The protocol is collusion-safe, as discussed in [12]; 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. The provider may still cache the ciphertext resulting from the most recent re-encryption.

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

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.

C. 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 by creating new re-encryption keys based on the manager-created hashes. 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 costly wireless 3G transfer.

VI. Evaluation and implementation of models

A. Qualitative cost comparison

The processing and communication costs of the transactions in the two proposed re-encryption models are shown in Table IV. 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 initial 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.

TABLE IV
Processing and communication costs of re-encryption models. 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 (mgr.)	-	$H+E$
Key re-generation (user)	-	-
Re-encryption (server)	-	BP
Re-encryption (manager)	$BP+E$	E
Computational costs		
Description	Proxy re-encryption	Cloud re-encryption
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

B. 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) [22], a porting of the PBC (Pairing-Based Cryptography Library) in C [23]. 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 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 V. 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 of memory. Note that these benchmark results were achieved using libraries that are not yet highly mature and optimized for a mobile platform.

TABLE V
Performance results obtained from the implementation.

Cryptographic task on 48-bit data block	Desktop (iMac)	Phone (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. with pairing, before re-encryption	4.9	128.9	2.9
Re-encryption time (ms), with pairing, from P_0 to P_1 keys	4.2	159.6	3.0
Decryption time (ms), using P_1 key, i.e. without pairing, after re-encryption	0.5	15.2	0.5

VII. Summary

A cryptographic protocol based on data re-encryption has been adapted to a cloud computing system model in order to gauge its viability in improving communication security and supporting highly scaleable and secure cloud computing applications serving an extremely large mobile device user population. A novel protocol based on data re-encryption has been proposed that leverages 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.

VIII. 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] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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

[16] 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.

[17] 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.

[18] P. Tysowski and M. A. Hasan, "Towards Secure Communication for Highly Scalable Mobile Applications in Cloud Computing Systems," Centre for Applied Cryptographic Research, University of Waterloo, Tech. Rep. CACR 2011-33, 2011.

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

[20] 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.

[21] 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.

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

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