

# No Shared Keys: Multi-user Searchable Data Encryption

Changyu Dong, Giovanni Russello, Naranker Dulay  
Department of Computing  
Imperial College London  
180 Queen's Gate, London, SW7 2AZ, UK  
{changyu.dong, g.russello, n.dulay}@imperial.ac.uk

## Abstract

*Current security mechanisms that provide data confidentiality require the servers and the administrators to be trusted and pose a risk for organisations that outsource their data management services. Encrypting sensitive data at the client side is the normal approach in this situation but makes it difficult to provide basic services such as keyword search. New cryptographic schemes have been proposed that support encrypted queries over encrypted data but all depend on a single set of secret keys, which implies single user access or sharing keys among multiple users. In this paper, we propose an encryption scheme where each authorised user in the system has his own keys to encrypt and decrypt data. The scheme supports keyword search which enables the server to return the encrypted data that satisfy an encrypted query without decrypting it. We provide two constructions of the scheme giving formal proofs of their security. We also report on the results of a prototype implementation.*

## 1 Introduction

Data growth is inevitable for nearly all organisations. According to Forrester Research, enterprise storage needs grow at 52 percent per year [5]. To reduce the increasing costs of storage management, many organizations choose to outsource their data storage to third party service providers. Recent research from TheInfoPro shows that nearly 20% of Fortune 1000 organizations outsource at least some portion of their storage management activities [12].

One of the biggest challenges raised by data storage outsourcing is security and trust. Business data is a valuable asset for many companies. While companies may trust a Storage Service Provider's (SSP) reliability, availability, fault-tolerance and performance, they cannot trust that an SSP is not going to use the data for other purposes, especially when the value of the data is high. Traditional access controls which are used to provide confidentiality are mostly designed for in-house services and depend greatly on the system itself to enforce authorization policies, effectively relying on a trusted infrastructure. In the absence of trust, traditional security models are no longer valid. Another common approach to provide data confidentiality is cryptography. Server side encryption is not appropriate when the server is not trusted. The client must encrypt the data before sending it to the SSP and later the encrypted data can be retrieved and decrypted by the client. This could ease a company's concern about data leakage, but introduces a new problem. Because the encrypted data is not meaningful to the servers, many useful data management functionalities are not possible. For example, if a client wants to retrieve documents or records containing certain keywords, how can this request be processed? Can we keep the data incomprehensible to servers and their administrators while efficiently retrieving the data? Consider the following scenarios:

**Scenario 1** *Company A is considering outsourcing its data processing centre to a service provider B. This will cut its annual IT cost by up to 25%. But the CIO is concerned about data security. The company's databases*

*contain valuable production data and customer information. It would be unacceptable if competitors got hold of the data. Administrative controls such as formal contracts, confidential agreements and continuous auditing provide a certain level of assurance, but the CIO would also like to encrypt the sensitive data and have fast searches over it.*

**Scenario 2** *Bob subscribes to a Personal Health Record service from company C. The service allows Bob to maintain his electronic medical records and share them with his doctors through a web interface. Bob wants to encrypt his records, ensuring the staff of company C will not be able to know what is inside.*

A trivial solution is to download all the data to the client's computer and decrypt it locally. This does not scale to large datasets. Recently, several innovative schemes have been proposed to address the above problems. The basic idea is to divide the cryptographic component between the client and server. The client performs the data encryption/decryption and manages the keys. The server processes search queries by carrying out some computation on the encrypted data. The server knows nothing about the keys or the plaintexts of the data nor the queries, but is still able to return the correct results.

These schemes also have an important limitation. The operations, e.g. encryption, decryption and query generation, more or less rely on some secret keys. This implies that the operations can only be executed by one user, or by a group of users who share the secret keys somehow. A single user is usually not an adequate assumption for data outsourcing. Perhaps the biggest problem for supporting multiple user access to encrypted data is key management. Sharing keys is generally not a good idea since it increases the risk of key exposure. In response to this, keys must be changed regularly. The keys must also be changed if a user is no longer qualified to access the data. However, changing keys may result in decrypting all the data and re-encrypting it using the new keys. For large datasets, this is not practical.

In this paper, we propose a scheme for multi-user searchable data encryption. Our scheme does not require a fully trusted server. The server can search an encrypted keyword on the encrypted data. More importantly each authorised user in the system has his own unique keys which simplifies key revocation and avoids data re-encryption. All the authorised users can insert encrypted data, decrypt the data inserted by other users and search encrypted data without knowing the other users' keys. The keys of one user can easily be revoked without affecting other users or the encrypted data at the server.

## **2 Related Work**

Recently, schemes have been developed to encrypt data on the client-side and enable server-side searches on encrypted data. [25] introduced the first practical scheme for searching on encrypted data. The scheme enables clients to perform searches on encrypted text without disclosing any information about the plaintext to the untrusted server. The untrusted server cannot learn the plaintext given only the ciphertext, it cannot search without the user's authorization, and it learns nothing more than the encrypted search results. The basic idea is to generate a keyed hash for the keywords and store this information inside the ciphertext. The server can search the keywords by recalculating and matching the hash value. [20] proposed a scheme to execute SQL queries over encrypted numeric data and is suitable for exact matches and also range queries. Its strategy is to store the encrypted numbers with some index information and to split the query into a query on the encrypted data processed by the untrusted server and a query on the returned result for post-processing results on the client. [7] presented a scheme for searches on encrypted data using a public key system that allows mail gateways to handle email based on whether certain keywords exist in the encrypted message. The application scenario is similar to [25], but the scheme uses identity-based encryption instead of symmetric ciphers. Using asymmetric keys allows multiple users to encrypt data using the public key, but only the user who has the secret key can search and decrypt the data. [18, 13] enable searches on encrypted data by constructing secure indexes. [26] proposed an elegant scheme for performing

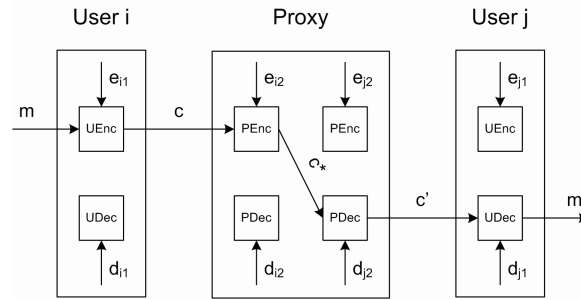
queries on encrypted data and also provided a secure index to speed up the queries. All the schemes above rely on secret keys however, which implies single user access or sharing keys among a group of users.

### 3 Multi-user Searchable Data Encryption Scheme: Basic Construction

The most significant characteristic of our searchable data encryption system is that we do not require sharing keys among the users. Each user has his own keys, which are unique in the system. This is achieved by using a *proxy encryption* scheme. In this section, we will introduce the basic construction of the multi-user searchable data encryption scheme which is built upon an RSA-based proxy encryption scheme. We will also formalize the notions of security and provide proofs later in this section.

#### 3.1 An RSA-based Proxy Encryption Scheme

The notion of proxy encryption was first introduced in [6]. In a proxy encryption scheme, a ciphertext encrypted by one key can be transformed by a proxy function into the corresponding ciphertext for another key without revealing any information about the keys and the plaintext. Proxy encryption schemes can be built on top of different cryptosystems such as El Gamal [14] and RSA [24]. There are many applications of proxy encryption, e.g. secure email lists [23], access control systems [3] and attribute based publishing of data [22]. A comprehensive study on proxy cryptography can be found in [21].



**Figure 1. Encryption/Decryption in Our RSA-based Proxy Encryption Scheme**

Our scheme uses an RSA-based proxy encryption scheme. Let's use  $\mathcal{E} = (IGen, UGen, UEnc, UDec, PEnc, PDec)$  to denote the proxy encryption scheme. Fig. 1 shows the encryption/decryption process in the proxy encryption scheme.

- *IGen* is the master key generation algorithm which is identical to the key generation algorithm in the standard RSA. It takes a security parameter  $k$  and generates  $(p, q, n, \phi(n), e, d)$ . *IGen* needs only to be run once at the beginning of the system setup. All the outputs except  $n$  must be kept secret. In the rest of the paper, we assume all arithmetic to be  $\text{mod } n$  unless stated otherwise.
- *UGen* is the algorithm for generating the key pair for the users and the proxy. For each user  $i$ , *UGen* takes the output of *IGen* and finds  $e_{i1}, e_{i2}, d_{i1}, d_{i2}$  such that  $e_{i1}e_{i2} \equiv e \text{ mod } \phi(n)$  and  $d_{i1}d_{i2} \equiv d \text{ mod } \phi(n)$ . This can be efficiently done. Take the  $e_{i1}, e_{i2}$  pair for example, we can pick  $e_{i1} < \phi(n)$  randomly, where  $e_{i1}$  is relatively prime to  $\phi(n)$ , i.e.  $\text{gcd}(e_{i1}, \phi(n)) = 1$ . Since  $e_{i1}x \equiv 1 \text{ mod } \phi(n)$  always has a solution, then  $e_{i2} \equiv ex \text{ mod } \phi(n)$  always satisfies  $e_{i1}e_{i2} \equiv e \text{ mod } \phi(n)$ . Note that knowing only  $a$  is not sufficient for solving the two variable equation  $ax \equiv y \text{ mod } n$ . Therefore by knowing only  $e_{i1}$  or  $e_{i2}$ , one cannot compute its counterpart ( $e_{i2}$  or  $e_{i1}$  respectively) and  $e$ . The user's key pair is  $(K_{uei}, K_{udi}) = (e_{i1}, d_{i1})$ . The proxy's corresponding key pair for the user  $i$  is  $(K_{pei}, K_{pdi}) = (e_{i2}, d_{i2})$ . The lower bound of the number of valid key pairs is  $\phi(\phi(n)) > \sqrt{\phi(n)}$ .

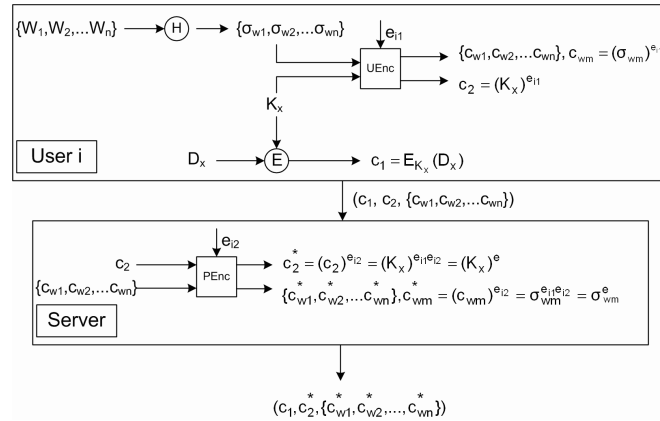
- *UEnc* is the algorithm for user encryption. For a message  $m$ , user  $i$  encrypts it using his encryption key  $K_{uei} = e_{i1}$ . The resulting ciphertext is  $c = m^{e_{i1}}$ .
- *UDec* is the algorithm for user decryption. When a user  $j$  receives a ciphertext  $c'$  from the proxy, he decrypts it using his decryption key  $K_{udj} = d_{j1}$ . He will be able to recover the plaintext  $m = (c')^{d_{j1}}$ .
- *PEnc* is the algorithm for proxy encryption. When the proxy receives a ciphertext  $c$  from user  $i$ , it re-encrypts it using the corresponding encryption key  $K_{pei} = e_{i2}$  as  $c^* = c^{e_{i2}}$ .
- *PDnc* is the algorithm for proxy decryption. Before sending the ciphertext to user  $j$ , the proxy decrypts it using the corresponding decryption key  $K_{pdj} = d_{j2}$  as  $c' = (c^*)^{d_{j2}}$ .

Note that in the system, for any user  $i$  and any user  $j$ ,  $e_{i1}e_{i2} \equiv e_{j1}e_{j2} \equiv e \pmod{\phi(n)}$  and  $d_{i1}d_{i2} \equiv d_{j1}d_{j2} \equiv d \pmod{\phi(n)}$ . Therefore  $c^* = c^{e_{i2}} = m^{e_{i1}e_{i2}} = m^e$ ,  $c' = (c^*)^{d_{j2}} = m^{ed_{j2}}$  and the user  $j$  can correctly decrypt  $c'$  because  $(c')^{d_{j1}} = m^{ed_{j2}d_{j1}} = m^{ed} = m$ .

In our system, we use a trusted key management server (KMS) to manage the keys. First, the KMS runs *IGen* to generate a master key pair  $(e, d)$  and publishes the only public parameter  $n$ . When a new user is enrolled into the system, the KMS runs *UGen* to generate a unique tuple  $((e_{i1}, d_{i1}), (e_{i2}, d_{i2}))$  and sends  $(e_{i1}, d_{i1})$  to the user and  $(e_{i2}, d_{i2})$  to the data server through secure channels. If the user is removed from the system at a later stage, the KMS can send a instruction to the data server to remove the key pair  $(e_{i2}, d_{i2})$  at the server side. We will see in the following sections, without the server side key pairs, the user cannot search and decrypt the data.

Although requiring a trusted KMS seems at odds with using an untrusted data storage service, we can argue that the KMS requires less resources and less management effort. Securing the KMS is much easier since a very limited amount of data needs to be protected and the KMS can be kept offline most of time.

### 3.2 Data Encryption



**Figure 2. Basic Data Encryption Scheme**

In our system, each data item  $D_x$  is associated with a set of searching keywords  $\{W_1, W_2, \dots, W_n\}_x$ . The encryption algorithm is shown in Fig. 2. The data item could be a document, an email in the email system, or a data cell in a database etc..

The data encryption is done at the client side using an semantically secure [19] symmetric encryption algorithm  $E$ . For each data item  $D_x$ , the user  $i$  picks a key  $K_x$  uniformly randomly from the key space of  $E$  and encrypts  $D_x$  under the key which generates a ciphertext  $c_1 = E_{K_x}(D_x)$ .  $K_x$  is then encrypted by the user's piece of RSA encryption key as  $c_2 = (K_x)^{e_{i1}}$ .

For each search keyword  $W_m$ , the client uses a hash function  $H$  to compute  $\sigma_{wm} = H(W_m)$  and computes  $c_{wm} = (\sigma_{wm})^{e_{i1}}$ . The client then sends the tuple  $(c_1, c_2, \{c_{w1}, c_{w2}, \dots, c_{wm}\})$  to the server.

After receiving the tuple, the server first computes  $c_2^* = c_2^{e_{i2}}$ . For each encrypted keyword  $c_{wm}$ , the server computes  $c_{wm}^* = c_{wm}^{e_{i2}}$ . The final cipher stored on the server is a tuple  $(c_1, c_2^*, \{c_{w1}^*, c_{w2}^*, \dots, c_{wm}^*\})$ .

### 3.3 Keyword Search

A user  $j$  may want to retrieve all the documents on the server which contain a keyword  $W$ . To do so,  $j$  first computes the hash value of the keyword  $\sigma = H(W)$ . Then  $j$  encrypts  $\sigma$  as  $Q = \sigma^{e_{j1}}$  and sends  $Q$  to the server.

The server re-encrypts  $Q$  as  $Q^* = Q^{e_{j2}}$ . Then it tests each ciphertext: in the encrypted keywords set  $\{c_{w1}^*, \dots, c_{wm}^*\}_x$ , if there exists a  $c_{wm}^*$  such that  $c_{wm}^* = Q^*$ , then adds this ciphertext into the result set.

Recall that  $e_{i1}e_{i2} \equiv e_{j1}e_{j2} \equiv e \pmod{\phi(n)}$ , so  $Q^* = (H(W))^e$  and  $c_{wm}^* = (H(W_m))^e$  are equal if and only if  $W = W_m$ .

If the server cannot find the corresponding key for the user  $j$ , it cannot correctly compute the searching keyword. Therefore an unauthorised user cannot perform searching on the data.

### 3.4 Data Decryption

If an authorised user  $j$  wants to retrieve  $D_x$ , the server gets the tuple  $(c_1, c_2^*, \{c_{w1}^*, c_{w2}^*, \dots, c_{wm}^*\})$  from the data storage, computes  $c'_2 = (c_2^*)^{d_{j2}}$  and sends  $c_1, c'_2$  to  $j$ . The user  $j$  then computes  $(c'_2)^{d_{j1}} = (c_2^*)^d = (K_x)^{ed} = K_x$  and can decrypt the data item  $D_x = E_{K_x}^{-1}(c_1)$ .

An unauthorised user cannot decrypt the data because the server does not have the corresponding proxy decryption key.

### 3.5 Attack Model

We focus the scope of our scheme on protecting data confidentiality, therefore we will not consider attacks on data integrity and availability which can be handled by other mechanisms. For the scheme, we assume that the KMS and the authorised users are fully trusted. We also assume they can properly protect their secrets, for example, the key pairs and the parameters for generating keys. The server is modeled as “honest-but-curious”, i.e. we trust it to correctly execute the instructions from the clients, but do not want it to access the plain data. An adversary  $Adv$  is an attacker (or a software agent) that gains privileged access to the data storage: either an outsider or a untrustworthy employee in the data centre. The adversary can also intercept the communications between clients and the server, but it is computationally bounded. In addition, the adversary is restricted to only perform passive attacks, i.e. attacks are based upon observed data. This restriction is reasonable because: (1) in most cases  $Adv$  is physically isolated from the users; (2) most communications between the clients and the server are one-round and initialised by the client, i.e. query-reply. The goal of the adversary is to gather direct or indirect information about the stored data.

### 3.6 Security Analysis

We now give the formal notions of security and proof of security for our system. Note that in the basic construction, the ciphertexts are encrypted by two different schemes. In such situations, the security of the whole system depends on the individual scheme. We assume that the symmetric key scheme is semantically secure, and will prove our proxy encryption scheme is One-Way secure.

**Definition 1.** Let  $\mathcal{E} = (IGen, UGen, UEnc, UDec, PEnc, PDec)$  be the proxy encryption scheme.  $\mathcal{E}$  is said to be One-Way secure against any PPT attacker  $\mathcal{A}$  if  $Succ_{\mathcal{A}, \mathcal{E}}$  is negligible.  $Succ_{\mathcal{A}, \mathcal{E}}$  is defined as follows:

$$Succ_{A,\mathcal{E}} = Pr \left[ m' = m \mid \begin{array}{l} (p, q, n, \phi(n), e, d) \leftarrow IGen(1^k), \\ (\mathcal{K}_u, \mathcal{K}_p) \leftarrow UGen(\phi(n), e, d), \\ m' \leftarrow \mathcal{A}(\mathcal{K}_p, n, m^\varepsilon), \varepsilon \in \mathcal{K}_u \end{array} \right]$$

Loosely speaking, the proxy encryption scheme is one-way secure if by knowing the public parameter  $n$ , all the key pairs on the server side, ciphertexts encrypted under an authorised user's encryption key and any information can be derived from above, e.g. intermediate ciphertexts calculated using the server side keys, but without knowing any key pairs in the authorised user key pair set  $\mathcal{K}_u$ , no PPT adversary can find the corresponding plaintext.

**Lemma 1.** *Under the RSA assumption, the proxy encryption scheme is One-Way secure against Adv.*

*Proof.* We will show that if Adv can break the proxy encryption scheme, i.e.  $Succ_{A,\mathcal{E}}$  is not negligible, then there is an attacker  $\mathcal{B}$  who can solve the RSA problem with non-negligible probability.

Given an RSA ciphertext  $c = m^e$  where the corresponding key pair is  $(e, d)$ , the goal of  $\mathcal{B}$  is to decrypt it, i.e. to find  $m$ .  $\mathcal{B}$  can pick  $x$  pairs of random primes  $\frac{n}{2} < (e_{\mathcal{B}}, d_{\mathcal{B}})_i < n - 2^{161}$ . The primes are relatively prime to  $\phi(n)$  because  $\frac{\phi(n)}{2} < (e_{\mathcal{B}}, d_{\mathcal{B}})_i < \phi(n)$ .  $\mathcal{B}$  then sends  $c, n, (e_{\mathcal{B}}, d_{\mathcal{B}})_i, i = 1, \dots, x$  to Adv.

Adv can compute  $c_1 = c^{e_{\mathcal{B}}}, c_2 = c_1^{d_{\mathcal{B}}}$ . Next we will show that  $c, c_1, c_2, n, (e_{\mathcal{B}}, d_{\mathcal{B}})_i, i = 1, \dots, x$  can correctly simulate adv's knowledge in the proxy encryption scheme. First we will show that  $c, c_1, c_2$  are valid ciphertexts for the proxy encryption scheme. The ciphertexts are valid if there exists a  $d'$  such that  $e_2^{d'} = m$ , i.e.  $ee_{\mathcal{B}}d_{\mathcal{B}}d' \equiv 1 \pmod{\phi(n)}$ . Because  $e_{\mathcal{B}}, d_{\mathcal{B}}$  are relatively prime to  $\phi(n)$ , we can always find  $y$  such that  $e_{\mathcal{B}}d_{\mathcal{B}}y \equiv 1 \pmod{\phi(n)}$ . Therefore there always exists  $d' \equiv dy \pmod{\phi(n)}$  such that  $ee_{\mathcal{B}}d_{\mathcal{B}}d' \equiv ee_{\mathcal{B}}d_{\mathcal{B}}dy \equiv (ed)(e_{\mathcal{B}}d_{\mathcal{B}}y) \equiv 1 \pmod{\phi(n)}$ . We also need to show that  $(e_{\mathcal{B}}, d_{\mathcal{B}})_i, i = 1, \dots, x$  are valid server side key pairs, this can be easily proved using the similar method as above therefore is omitted.

Now with the message from  $\mathcal{B}$ , Adv can find  $m$  with probability  $Succ_{A,\mathcal{E}}$  and returns the result to  $\mathcal{B}$ . This means  $\mathcal{B}$  can solve the RSA problem with non-negligible probability  $Succ_{A,\mathcal{E}}$ , which contradicts the RSA assumption.  $\square$

**Theorem 1.** *The basic construction is One-Way secure against Adv.*

This is quite straightforward. The ciphertext is encrypted disjointedly by two encryption schemes. The symmetric encryption scheme is semantically secure, i.e. ciphertext indistinguishable, which implies it is One-Way secure against Adv. Since the proxy encryption scheme has been proved to be One-Way secure, overall, the basic construction is One-Way secure against Adv.

One-Way secure is sufficient to protect a data item, since an adversary cannot recover the symmetric key and then decrypt the data item. But it does leak some information about the keywords. Because the proxy encryption is deterministic, the ciphertexts of keywords are not indistinguishable. All the occurrences of the same keyword generate the same ciphertext. The adversary can make inferences from the keyword distribution by observing the encrypted data service.

In the following section, we will show a enhanced scheme which is semantically secure and makes the above attack impossible.

## 4 Enhanced Construction

The problem with the basic construction comes from the fact that the keyword encryption is not semantically secure. Using some probabilistic padding schemes can solve the problem, but then the encrypted keywords are no longer searchable. In this section, we will show a enhanced construction with a new keyword encryption scheme which is both semantically secure and searchable.

#### 4.1 Keyword Encryption Scheme

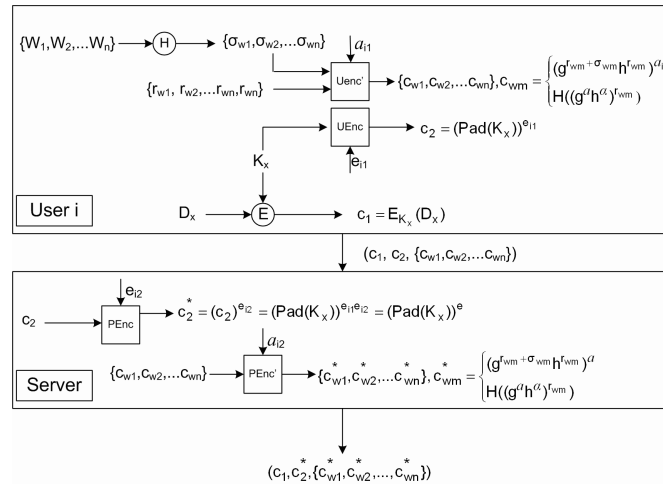
In the new construction, to avoid the problem discussed in section 3.6, the keywords are no longer encrypted under the proxy encryption scheme. Instead, we encrypt each keyword as a non-interactive zero-knowledge proof style witness and put the witness in the data server. An additional key pair is generated for encrypting the keywords and in the queries. The new keyword encryption scheme is based on Discrete Logarithms.

Let  $\mathcal{E}' = (IGen', UGen', UEnc', PEnc')$  denote the keyword encryption scheme.

- $IGen'$  is the algorithm for generating the public parameters and the master key. It takes a security parameter  $k$  and generates  $\{p', q', g, x, h, a, g^a h^a\}$ .  $p'$  and  $q'$  are two large prime numbers such that  $q'$  divides  $p' - 1$ .  $g$  is a generator of  $G_{q'}$ , the unique order- $q'$  subgroup of  $Z_{p'}^*$ .  $h \equiv g^x \bmod p'$  where  $x$  is chosen uniformly randomly from  $Z_{q'}$ .  $a$  is also a random number from  $Z_{q'}$ .  $p', q', g, h, g^a h^a$  are publicised and  $x, a$  must be kept secret. The reason why we publish  $g^a h^a$  instead of  $g^a$  is that if  $g^a$  is available to the adversary, then it can generate search queries of any chosen keywords.
- $UGen'$  is the algorithm for generating the key pairs for the users and the proxy. For a user  $i$ , it finds  $a_{i1} a_{i2} \equiv a \bmod q'$ . The user's keyword encryption key is  $a_{i1}$ , and the proxy's share is  $a_{i2}$ . The number of key pairs is at least  $\phi(q') = q' - 1$ .
- $UEnc'$  is the client-side encryption algorithm.
- $PEnc'$  is the server-side encryption algorithm.

Note that there is no decryption algorithm for this keyword encryption scheme. This is because the ciphertexts of the keywords are only used for testing whether there is a match and do not need to be decrypted.

#### 4.2 Data Encryption/Decryption



**Figure 3. Data Encryption Algorithm 2**

The new encryption scheme is shown in Fig. 3. The data item encryption/decryption is the same as in the basic construction. To make the ciphertexts of the symmetric keys more secure, we modify the user encryption/decryption algorithms in the RSA proxy encryption scheme to use OAEP (Optimal Asymmetric Encryption Padding) [4], a probabilistic padding scheme. RSA-OAEP has been proved to be IND-CCA2 (Indistinguishable

Adaptive Chosen Ciphertext Attack) secure [15]. Now the symmetric key is encrypted by the user's piece of the RSA encryption key as  $c_2 = (Pad(K_x))^{e_{i1}}$ . The server side proxy encryption/decryption algorithms remain the same, i.e. modular exponentiation.

The keywords are now processed as follows: for each keyword  $W_m$ , the user  $i$  computes  $\sigma_{wm} = H(W_m)$  using a hash function  $H$ . The user also picks a random number  $r_{wm} \in Z_{q'}$  and computes  $c_{wm,1} = (g^{r_{wm} + \sigma_{wm}} h^{r_{wm}})^{a_{i1}} \mod p'$ ,  $c_{wm,2} = H((g^a h^a)^{r_{wm}})$ , where  $g, h, g^a h^a, p'$  are public parameters in the system and  $a_{i1}$  is the user's keyword encryption key. The user then sends the tuple  $(c_1, c_2, \{c_{w1}, c_{w2}, \dots, c_{wn}\})$  to the server, where  $c_{wm}$  is the tuple  $(c_{wm,1}, c_{wm,2})$ .

The server re-encrypts the data encryption key as in the basic construction. Then it processes the keywords information. For each  $c_{wm}$ , the server computes  $c_{wm,1}^* = c_{wm,1}^{a_{i2}} = (g^{r_{wm} + \sigma_{wm}} h^{r_{wm}})^{a_{i1} a_{i2}} = (g^{r_{wm} + \sigma_{wm}} h^{r_{wm}})^a \mod p'$ ,  $c_{wm,2}^* = c_{wm,2}$ . The final cipher stored is a tuple  $(c_1, c_2^*, \{c_{w1}^*, c_{w2}^*, \dots, c_{wn}^*\})$ , where  $c_{wm}^* = (c_{wm,1}^*, c_{wm,2}^*)$ .

### 4.3 Keyword Search

To search for a keyword  $W$ , the user  $j$  computes  $\sigma = H(W)$ . The user then computes the encrypted query  $Q = g^{(-\sigma)a_{j1}} \mod p'$  and sends it to the server. The server computes  $Q' = Q^{a_{j2}} \mod p' = g^{(-\sigma)a} \mod p'$ . For each  $c_{wm}^*$ , the server computes:

$$\begin{aligned} y_1 &= c_{wm,1}^* Q' = (g^{r_{wm} + \sigma_{wm}} h^{r_{wm}})^a g^{(-\sigma)a} = (g^{ar_{wm} + a\sigma_{wm}} h^{ar_{wm}}) g^{(-a\sigma)} \mod p' \\ y_2 &= H(y_1) \end{aligned}$$

We can see that if  $a\sigma_{wm} - a\sigma = 0$ , i.e.,  $W_m = W$ , then  $y_1 = (g^{ar_{wm}} h^{ar_{wm}}) = (g^a h^a)^{r_{wm}} \mod p'$  and therefore  $y_2 = H((g^a h^a)^{r_{wm}}) = c_{wm,2}^*$ . Then by comparing  $y_2$  and  $c_{wm,2}^*$ , the server can decide whether the keyword matches the query.

### 4.4 Security Analysis

We first prove that the keyword encryption is semantically secure. Semantic security means that the ciphertexts are indistinguishable to the adversary, therefore the adversary learns nothing by looking at the ciphertext.

**Lemma 2.** Let the keyword encryption  $\mathcal{KE} = (Pub\_para, Sec\_para, \mathcal{K}_u, \mathcal{K}_p, Enc)$  where  $Pub\_para$  is the public parameter set,  $Sec\_para$  is the secret parameter set,  $\mathcal{K}_u, \mathcal{K}_p$  are the user and proxy key sets respectively,  $Enc, Dec$  are the encryption/decryption algorithms. It is semantically secure against any PPT attacker, i.e.  $Succ_{\mathcal{A}, \mathcal{KE}}$  is negligible, where

$$Succ_{\mathcal{A}, \mathcal{KE}} = Pr \left[ b' = b \mid \begin{array}{l} m_0, m_1 \in \{0, 1\}^l, \\ b \xleftarrow{R} \{0, 1\}, \\ b' \leftarrow \mathcal{A}(Pub\_para, \mathcal{K}_p, Enc_k(m_b)), k \in \mathcal{K}_u \end{array} \right] - \frac{1}{2}$$

*Proof.* The ciphertext of a keyword  $m_b$  in the form of  $c_{m_b} = ((g^{r_{m_b} + \sigma_{m_b}} h^{r_{m_b}})^{a_{i1}}, H((g^a h^a)^{r_{m_b}}))$ . It's easy to see that if  $r_{m_b}$  is selected uniformly randomly from  $Z_{q'}$ , then  $g^{r_{m_b} + \sigma_{m_b}} h^{r_{m_b}}$  is distributed uniformly in  $G_{q'}$ . We will show that if  $Succ_{\mathcal{A}, \mathcal{KE}}$  is non-negligible, then there is an attacker  $\mathcal{B}$  who can win the following game with a non-negligible probability  $Succ_{\mathcal{B}, \mathcal{C}}$ , which is impossible.

$$Succ_{\mathcal{B}, \mathcal{C}} = Pr \left[ b' = b \mid \begin{array}{l} m_0, m_1 \in \{0, 1\}^l, \\ b \xleftarrow{R} \{0, 1\}, r \xleftarrow{R} Z_{q'}, \sigma_{m_b} = H(m_b) \\ b' \leftarrow \mathcal{A}(p', q', g, h, H, g^{r + \sigma_{m_b}} h^r) \end{array} \right] - \frac{1}{2}$$



$\mathcal{B}$  first sends  $m_0, m_1$  to the encryption oracle and receives  $g^{r+\sigma_{m_b}} h^r$ . Then it chooses a random number  $a \in \mathbb{Z}_{q'}$  and generates  $n$  pairs of  $(a_{i1}, a_{i2})$  such that  $a_{i1}a_{i2} \equiv a \pmod{p'}$ . It also computes  $\sigma_{m_0} = H(m_0)$  and  $\theta = g^{r+\sigma_{m_b}} h^r g^{-\sigma_{m_0}}$ , it is clear that  $\Pr[\theta = g^r h^r] = \frac{1}{2}$ .

Then  $\mathcal{B}$  sends  $(m_0, m_1, p', q', g, h, g^a h^a, (g^{r+\sigma_{m_b}} h^r)^{a_{11}}, (g^{r+\sigma_{m_b}} h^r)^a, H(\theta^a), a_{12}, \dots, a_{n2})$  to  $\mathcal{A}$ . If  $\theta = g^r h^r$ , then  $\mathcal{A}$  can output  $b' = b$  with probability  $\text{Succ}_{\mathcal{A}, \mathcal{KE}}$ . Therefore the probability of  $\mathcal{B}$  to win the game is  $\text{Succ}_{\mathcal{B}, \mathcal{C}} = \text{Succ}_{\mathcal{A}, \mathcal{KE}}/2$ , which is non-negligible.  $\square$

The semantically secure definition for searchable encryption is tricky because we must consider the information leaked by the queries. The queries enable the adversary to build search patterns and launch statistical attacks. Although the queries are encrypted, the adversary can still learn some information. Here we adapt the definition from [13] and prove our scheme is non-adaptive semantically secure. Informally, non-adaptive semantic security means that given two non-adaptively generated query histories with the same length and outcome, no PPT adversary can distinguish one from another with non-negligible probability. Non-adaptive means the adversary cannot adaptively choose queries based on the prior results. This is acceptable because in our setting, only the authorised user can generate queries.

We first introduce some notions to be used in the definition.  $\Delta = \{w_1, \dots, w_d\}$  is the dictionary and  $\mathcal{P}(\Delta)$  is the power set of  $\Delta$ , i.e. the set of all possible documents.  $\mathcal{P}(\mathcal{P}(\Delta))$  is the power set of  $\mathcal{P}(\Delta)$ ,  $\mathcal{D} \in \mathcal{P}(\mathcal{P}(\Delta))$  is an arbitrary subset of  $\mathcal{P}(\Delta)$  which contains  $n$  documents  $(D_1, \dots, D_n)$ . For each document, it is associated with a unique identifier  $\text{id}(D)$ , and a set of keywords in the documents  $\text{kw}(D) \subseteq D$ . The outcome of a search query is denoted by  $\mathcal{D}(w)$  which is the set of document identifiers such that  $\forall \text{id}(D) \in \mathcal{D}(w), D \in \mathcal{D} \wedge w \in \text{kw}(D)$ , i.e. all the document in the documents set which contain the keyword.

**Definition 2 (History).** A history  $H_q \in \mathcal{P}(\mathcal{P}(\Delta)) \times \Delta^q$  is an interaction between a client and a server over  $q$  queries. The partial history  $H_q^t \in \mathcal{P}(\mathcal{P}(\Delta)) \times \Delta^q$  of a given history  $H_q = (\mathcal{D}, w_1, \dots, w_q)$ , is the sequence  $H_q^t = (\mathcal{D}, w_1, \dots, w_t)$ , where  $t \leq q$ .

During the interaction, the adversary cannot see directly the documents, keywords and queries, but only their encrypted versions. Let  $E$  be the symmetric key encryption scheme,  $\mathcal{E}$  be the proxy encryption scheme and  $\mathcal{KE}$  be the keyword encryption scheme,  $Q$  be the encrypted query, the view of the adversary is then defined as:

**Definition 3 (View).** Given a documents set  $\mathcal{D}$  with  $n$  documents and a history over  $q$  queries  $H_q = (\mathcal{D}, w_1, \dots, w_q)$ , an adversary's view of  $H_q$  is defined as  $V(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), E_{k_1}(D_1), \dots, E_{k_n}(D_n), \mathcal{E}(k_1), \dots, \mathcal{E}(k_n), \mathcal{KE}(\text{kw}(D_1)), \dots, \mathcal{KE}(\text{kw}(D_n)), Q_1, \dots, Q_q)$ . The partial view  $V^t(H_q)$  of a history  $H_q$  is  $V^t(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), E_{k_1}(D_1), \dots, E_{k_n}(D_n), \mathcal{E}(k_1), \dots, \mathcal{E}(k_n), \mathcal{KE}(\text{kw}(D_1)), \dots, \mathcal{KE}(\text{kw}(D_n)), Q_1, \dots, Q_t)$ , where  $t \leq q$ .

As we have stated above, we must also consider the information leaked by the ciphertexts stored on the server but also the search outcomes. The information we have to leak is captured by trace:

**Definition 4 (Trace).** Given a documents set  $\mathcal{D}$  with  $n$  documents and a history over  $q$  queries  $H_q$ , the trace of  $H_q$  is defined as  $\text{Tr}(H_q) = (\text{id}(D_1), \dots, \text{id}(D_n), |D_1|, \dots, |D_n|, |\text{kw}(D_1)|, \dots, |\text{kw}(D_n)|, \mathcal{D}(w_1), \dots, \mathcal{D}(w_n), \Pi_q)$ .  $\Pi_q$  is the search pattern over the history which is a symmetric binary matrix where  $\Pi_q[i, j] = 1$  if  $w_i = w_j$ , and  $\Pi_q[i, j] = 0$  otherwise, for  $1 \leq i, j \leq q$ .

**Definition 5 (Non-Adaptive Semantic Security).** Our searchable data encryption is Non-Adaptive Semantically Secure if for all  $q \in \mathbb{N}$ , for all  $(H_0, H_1)$  which are histories over  $q$  queries and  $\text{Tr}(H_0) = \text{Tr}(H_1)$ , and any PPT adversary  $\mathcal{A}$ ,  $\text{Succ}_{\mathcal{A}}$  is negligible:

$$\text{Succ}_{\mathcal{A}} = \Pr \left[ b' = b \mid \begin{array}{l} \text{Pub}_{\text{para}}, \text{Sec}_{\text{para}}, \mathcal{K}_u, \mathcal{K}_p \leftarrow \text{SETUP}(1^k), \\ H_0, H_1 \in \mathcal{P}(\mathcal{P}(\Delta)) \times \Delta^q, \\ b \xleftarrow{R} \{0, 1\}, \\ b' \leftarrow \mathcal{A}(\text{Pub}_{\text{para}}, \mathcal{K}_p, V(H_b)) \end{array} \right] - \frac{1}{2}$$

**Theorem 2.** *The enhanced construction is non-adaptive semantically secure.*

*Proof.* For  $q = 0$ , i.e. when there is no query, the view of the adversary is  $V(H_b) = (id(D_1), \dots, id(D_n), E_{k_1}(D_1), \dots, E_{k_n}(D_n), \mathcal{E}(k_1), \dots, \mathcal{E}(k_n), \mathcal{KE}(kw(D_1)), \dots, \mathcal{KE}(kw(D_n)))$ . Because  $Tr(H_0) = Tr(H_1)$ , the adversary cannot distinguish the two histories by the document identifiers. Then for the other parts of the view:

**Encrypted documents:** The adversary cannot distinguish because  $E$  is semantically secure.

**Encrypted symmetric keys:**  $\mathcal{E}$  is based on RSA-OAEP which is IND-CCA2 secure, we can prove  $\mathcal{E}$  is also IND-CCA2 secure according to [21]. Therefore is also indistinguishable.

**Encrypted keywords:** We have proved they are indistinguishable to the adversary in lemma 2.

For  $q \geq 0$ , we must prove that given additional information, i.e.  $q$  queries  $(Q_1, \dots, Q_q)$ , the two histories are still indistinguishable. Because  $Tr(H_0) = Tr(H_1)$ , we can reduce the problem to distinguish two sequence of distinct queries:  $(Q_{01}, \dots, Q_{0m}), (Q_{11}, \dots, Q_{1m}), m < q$ . For each  $Q_{ij}, i \in 0, 1, 1 \leq j \leq m$ , it is a pseudorandom number  $g^{a_1 H(w_{ij})} \bmod p'$ . Therefore the queries are not distinguishable. □

## 5 Other Considerations

### 5.1 User Revocation

Access to encrypted data involves both client-side and server-side keys. So revoking a user's access is quite simple. The KMS can send an instruction to the server to let it remove the user's corresponding keys on the server side. After the keys have been removed, the user cannot access the data unless the KMS generates new keys for him. Even a revoked user can masquerade as an authorised user, his requests cannot be processed correctly if he does not know the authorised user's keys.

### 5.2 User Authentication and Secure Channel from RSA Keys

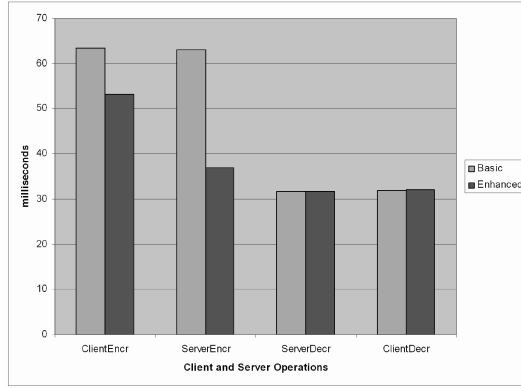
Each authorised user has his own RSA key pair  $(e_{i1}, d_{i1})$  and the server holds the corresponding key pair  $(e_{i2}, d_{i2})$ . Because  $e_{i1}d_{i1}e_{i2}d_{i2} \equiv ed \equiv 1 \bmod \phi(n)$ ,  $k_1 = e_{i1}d_{i1}$  and  $k_2 = e_{i2}d_{i2}$  form another RSA key pair. This key pair can be used for public key based mutual authentication and to establish a secure channel e.g. SSL. This adds another layer of protection against unauthorised users.

### 5.3 Collusion Attacks

The main concern with proxy encryption schemes comes from a collusion attack. If a user colludes with adversary  $Adv$ , who knows all the server side keys, they can easily recover the master keys by combining their keys. Although some work has been done in [3] using bilinear map to prevent the colluded parties from recovering the master key, the colluded parties are still able to decrypt the ciphertext with a weak secret they can recover. Theoretically, the design of collusion-resistant proxy encryption schemes is an open problem. But in practice, we can lower the risk to an acceptable level by implementing other mechanisms. For example, we can limit the access to the keys by using tamper-proof devices. We can also split the master keys into multiple shares and introduce additional servers, making collusion more difficult. Monitoring and auditing to detect collusion can also help to mitigate the risk.

## 6 Implementation and Performance

In this section we discuss the performance of a prototype implementation of both the basic and the enhanced construction of the encryption scheme presented in this paper.



**Figure 4. Performance of Data Encryption/Decryption Operations**

The prototype is implemented in Java using the packages provided in the standard Java 1.5 distribution. We use our encryption scheme to encrypt a single table database. We chose AES as the symmetric cipher which encrypts the actual data and SHA-1 as the hash function. For the RSA-based proxy encryption scheme, we used 1024-bit keys. For the keyword encryption scheme,  $q'$  was 160-bit and  $p'$  was 1024-bit. The tests were executed on a Intel Pentium IV 3.2 GHz (dual core) with 1 GB of RAM.

The first evaluation consists of measuring for each scheme the execution time of the following operations:

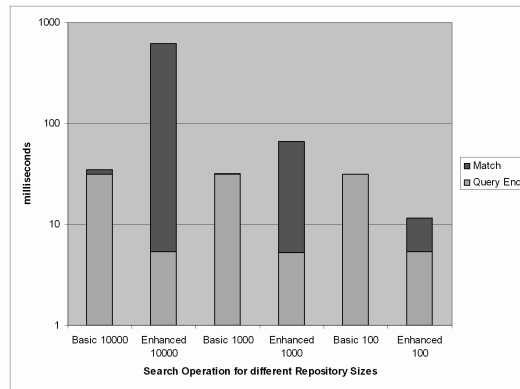
- Client Encryption: that consists of encrypting a data item using the symmetric cipher, encrypting the symmetric key and encrypting the keywords.
- Server Encryption: re-encryption of the symmetric key and the keywords using the server side keys.
- Server Decryption: pre-decryption of the symmetric key.
- Client Decryption: decryption of the symmetric key and the data item.

The graph in Fig. 4 shows the performance for the execution of encryption and decryption operations for each construction. The time in the Y-axis is given in milliseconds. The graph provides the average time for 10,000 executions. The data item we used in the experiments was a 16-byte string with one associated keyword.

The result shows that the enhanced construction has better performance than the basic construction in encryption. Since the data encryption and key encryption are nearly identical in both constructions, the difference is due to the fact that they encrypt the keywords using different schemes. The enhanced construction encrypts the keywords using a Discrete Logarithm based scheme and the basic construction uses an RSA-based scheme. The exponent used in the DL scheme is smaller than that of the RSA scheme, therefore the keyword encryption of the DL scheme is faster than the RSA scheme. The decryption part of both constructions are almost the same, so we can see from the figure that the two constructions have nearly the same performance in decryption.

We also measured the time for processing a search query on the server side in both constructions, the result is shown in Fig. 5. Processing a search query involves two operations: query re-encryption (Query Encr) and matching (Match). The graph shows the time (in milliseconds) for a search operation (the time scale is logarithmic) executed on several databases with different sizes in both constructions. We used three databases containing 100, 1,000 and 10,000 keywords each.

The graph shows that for the basic construction, the query encryption dominates the overall searching time. This is easy to understand since the matching operation in the basic construction is simply string comparison. Therefore the size of the database has little effect on the searching time in the basic construction. In contrast,



**Figure 5. Performance of Searching Operations**

the time spent on the matching operation is much more significant in the enhanced construction. And when the database becomes large, the time increases linearly. As a result, the basic construction has better performance than the enhanced construction when searching large databases.

## 7 Conclusion and Future Work

In this paper, we presented a new data encryption scheme that does not require a trusted data server. In the scheme the server can perform searches on the encrypted data without knowing the plaintext or the decryption keys. Unlike previous searchable data encryption schemes that require a shared key for multi-user access, each user in our system has a unique set of keys. The data encrypted by one user can be correctly decrypted by all the authorised users in the system. Moreover the keys can be easily revoked without any overhead, i.e. without having to re-encrypt the stored data.

We provided two constructions for the scheme built on top of proxy encryption schemes. For each construction, we gave the formal definitions and proofs of security. We also implemented them in Java and compared the performance.

One aspect of our future work is to investigate and integrate our scheme with Private Information Retrieval (PIR) schemes. PIR schemes [11, 8, 17] allow a user to retrieve some items from a database without revealing to the database which items were queried. A weakness of our scheme is that it allows statistical attacks on the queries. By combining PIR techniques, we could potentially make our scheme more secure.

Secure indexes [18, 10] is another promising technique that is used to improve the performance and decrease the storage overhead of searchable encryption schemes. We will investigate current schemes and develop a new index scheme for the multi-user system.

An interesting possible application for our encryption schemes is in the Tuple Space Model (TSM) [16]. The TSM supports a matching model where queries for searching a tuple are specified using either the exact value or a wildcard. The encryption and searching algorithms are well suited for such a model.

## References

- [1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.
- [2] R. Anderson. Two remarks on public-key cryptology. Technical Report UCAM-CL-TR-549, Univesity of Cambridge, 2002.

- [3] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*. The Internet Society, 2005.
- [4] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.
- [5] J. Blackwood. Is storage outsourcing a viable alternative?  
<http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2851289,00.html>.
- [6] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.
- [7] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
- [8] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.
- [9] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.
- [10] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, 2005.
- [11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.
- [12] D. Connor. Storage outsourcing on the rise.  
<http://www.networkworld.com/news/2007/012207-storage-outsourcing-rises.html>, 2007.
- [13] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 79–88. ACM, 2006.
- [14] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- [15] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2001.
- [16] D. Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.
- [17] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer, 2005.
- [18] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. <http://eprint.iacr.org/2003/216/>.
- [19] O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.
- [20] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *SIGMOD Conference*, pages 216–227. ACM, 2002.
- [21] A.-A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*. The Internet Society, 2003.
- [22] A. Kapadia, P. P. Tsang, and S. W. Smith. Attribute-based publishing with hidden credentials and hidden policies. In *NDSS*. The Internet Society, 2007.
- [23] H. Khurana, A. J. Slagell, and R. Bonilla. Sels: a secure e-mail list service. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 306–313. ACM, 2005.
- [24] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [25] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
- [26] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 479–495. Springer, 2006.