Privacy-enhanced keyword search in clouds

Miao Zhou
University of Wollongong, mz775@uowmail.edu.au

Yi Mu
University of Wollongong, ymu@uow.edu.au

Willy Susilo
University of Wollongong, wsusilo@uow.edu.au

Man Ho Allen Au
University of Wollongong, aau@uow.edu.au

Privacy-enhanced keyword search in clouds

Abstract

The advent of cloud computing has dramatically changed the IT scene, as it offers cost savings and improvements to major operations. Nevertheless, the major obstacle relies on the effort on how to secure sensitive data files that are outsourced to the cloud environment. To ensure confidentiality, the sensitive data are usually encrypted prior to being outsourced. Nevertheless, effective data utilization remains a challenging task and there is a clear need for a secure and efficient searching mechanism over the encrypted data in the cloud, to increase the us-ability of the secure cloud environment. Unfortunately, existing work in the area of secure searching in the outsourcing scenario usually incur high computational complexity, which makes the approach impractical. In this paper, we take one step ahead by proposing an efficient keyword search scheme for cloud computing. Our solution is very lightweight, and it enables efficient multi-user keyword search over outsourced data files in the cloud environment, without leaking any private information about both the data owner and users in the search query. We formally define the security requirements and prove that our scheme is secure under a simple assumption in the standard model.

Keywords

era2015, keyword, enhanced, clouds, privacy, search

Disciplines

Engineering | Science and Technology Studies

Publication Details

Zhou, M., Mu, Y., Susilo, W. & Au, M. (2013). Privacy-enhanced keyword search in clouds. 12th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (pp. 447-451). Canada: IEEE.

Privacy-Enhanced Keyword Search in Clouds

Miao Zhou, Yi Mu, Willy Susilo, Man Ho Au

Abstract—**The advent of cloud computing has dramatically changed the IT scene, as it offers cost savings and improvements to major operations. Nevertheless, the major obstacle relies on the effort on how to secure sensitive data files that are outsourced to the cloud environment. To ensure confidentiality, the sensitive data are usually encrypted prior to being outsourced. Nevertheless, effective data utilization remains a challenging task and there is a clear need for a secure and efficient searching mechanism over the encrypted data in the cloud, to increase the usability of the secure cloud environment. Unfortunately, existing works in the area of secure searching in the outsourcing scenario usually incur high computational complexity, which makes the approach impractical. In this paper, we take a further step by proposing an efficient keyword search scheme for cloud computing. Our solution is very simple, and it enables efficient multi-user keyword search over outsourced data files in the cloud environment, without leaking any private information about either the data owner or users in the search query. We formally define the security requirements and prove that our scheme is secure under a simple assumption in the standard model.**

Index Terms—**Keyword search, cloud computing lightweight.**

I. INTRODUCTION

Due to its low cost, robustness and flexibility, cloud computing changes the way entities manage their data and offers individuals and companies with affordable storage, professional maintenance and adjustable space. Among the four cloud computing deployed models that includes: public, private, community and hybrid, public cloud where the outsourced resources can be accessed by the general public has gain a dramatic growth. By using a public cloud, a variety of users could access or share information that are stored in the cloud, independent of their different locations. Meanwhile, it also makes effective data utilization a challenging task as the outsourced data are usually in the encrypted form.

To set the scene, let us consider the following scenario. Let us consider a user Alice who uses a public cloud to store her personal data files such as family photos, blogs and working documents. To prevent the cloud server from learning the contents, she encrypts all her files prior to sending her data to the cloud. Once a while, she would like to access her files from different devices, such as a Boxee Box, an Apple TV or even her iPhone. Naturally, Alice would not remember all the contents that she has stored in the cloud. Therefore, there is a need for efficient searching over her outsourced files using the appropriate keywords. Since some of her mobile devices are only equipped with limited computational power, the searching mechanism should be very efficient, and it should ideally avoid using the relatively expensive techniques in public key cryptography, such as bilinear pairings. In other some cases, she would also like to share some of the files with her family and friends. For example, data files with labels "family" and "friends" from Alice would be accessible by her family member and friends, respectively. However, this requires Alice to define whom her family and friends are and requires the cloud server to enforce access control. This requirement may burden the regular users with the required expensive operations, and it may also reveal some information with regards to Alice's social networks.

In other words, we are looking for a practical scheme that provides:

- an efficient data search, and
- a simple access control.

a) Overview of Our Approach.: It turns out that a simple and straightforward approach could fit Alice's requirements. Before outsourcing each data file, Alice attaches a 'hidden index' h related to a certain keyword w to it. The hidden index h is computed as $H(w)$ for a hash function H using a keyword w. This can be easily extended to the multi-keyword case where each file F is attached with several hidden indexes for the relevant keywords. In order to search all files related to a keyword w^*, Alice compute $h^* = H(w^*)$ and sends h^* to the cloud, who returns all the data files attached with hidden index h^*. This idea can be used for simple access control as well. For instance, data files to be shared with Alice's friends could be attached be a hidden index with $w =$"friends". Alice's friends could then access those files using "friends" as the searching keyword. In this case a keyword plays the role of a password.

Unfortunately, this approach is inadequate in terms of the keyword search as well as a simple access control. Firstly, it leaks some information about the keyword to the cloud since the cloud can guess the keywords by testing whether the hash of it matches with the hidden index. Secondly, the keyword are human-memorable and is thus suspectable to the guessing attack. This is thus unsuitable even for a simple access control purpose.

One possible way to deal with these vulnerabilities is to generate a random value t_i for each possible keyword w_i. Since t_i is completely random, the scheme will not be vulnerable to the guessing attack. The drawback is apparent. Alice is required to build up a look up table which links every keyword to its corresponding random number. Firstly, the table could be large if Alice would set the file name as keyword (which is natural since this would allow her to search using the file name). Secondly, Alice has to keep an up-to-date copy of the table in all her devices.

Finally, we tackle the above issue by the use of the pseudo-random function (PRF). Instead of generating a random value t_i for each possible keyword w_i, Alice computes t_i as the output of the pseudo-random function with input w_i and a secret seed s. The value of s is kept secret and is stored in all Alice's devices. Indeed, the issue of storing the look up table has been reduced to just storing merely one secret seed s.

We call our system simple privacy-enhanced keyword search in clouds (*SPEKS*) to emphasize its simplicity. The term privacy-enhanced is used to reflect the privacy guarantee about our scheme. The cloud server can still tell if two data files share the same keyword. Exact security guarantee provided by our system will be formalized in subsequent sections.

b) Paper Organization.: The rest of the paper will be organized as follows. In Section II, we present related work. Formalization of security requirements are developed in In Section III. Our construction and security analysis are given in Section IV. In Section V, we estimate the performance of our scheme. Finally, we conclude our paper in Section VI.

II. RELATED WORK

Existing works close to ours can be found in the areas of "searching with privacy" and "searching on private-key-encrypted data". In theory, the classical work of Goldreich and Ostrovsky [9] on oblivious RAMs can resolve the problem of doing (private) searches on remote encrypted data, where oblivious RAMs hide all information about the RAM use from a remote and potentially malicious server with a poly-logarithmic overhead in all parameters (including computation and communication). Although their scheme is asymptotically efficient and nearly optimal, it does not appear to be efficient in practice as large constants are hidden in the big-$\mathcal{O}$ notation.

In an effort to reduce the round complexity associated with oblivious RAMs, Song, Wagner and Perrig [15] presented a solution for searchable encryption and after that how to do keyword searches on encrypted data efficiently was raised. In [15], they achieved searchable encryption by constructing a special two-layered encryption for each word. Given a trapdoor, the server can strip the outer layer and assert whether the inner layer is the correct form. The limitations in this construction are as follows. First, it is not compatible with existing file encryption schemes and a specific encryption method must be used; second, while the construction is proven to be a secure encryption scheme, it is not proven to be a secure searchable encryption scheme and the distribution of the underlying plaintexts is vulnerable to statistical attacks, that is, their approach may leak the locations of the keyword in a file; finally, searching is linear in the length of the document collection.

The above limitations are addressed by Goh [10], Chang and Mitzenmacher [6] and also Curtmola, Garay, Kamara and Ostrovsky [7], etc. In [10], they built an index of keywords for each file using a Bloom filter with pseudo-random functions used as hash functions. One inherent problem with this Bloom-filter-based approach is that Bloom filters can induce false positives, which would potentially cause mobile users to download extra files not containing the keyword. In [6], Chang and Mitzenmacher achieved the notion of security to IND2-CKA for chosen keyword attack, except that it also tries to guarantee that the trapdoors do not leak any information about the words being queried. In [7], they proposed a multi-user construction that is efficient on server's side, however every node in the link list has to be augmented with information about the file index of the next node.

In a different direction, Boneh, di Crescenzo, Ostvrosky and Persiano [3] and Boneh, Kuchilevitz, Ostvrosky and Skeith [4] studied the problem of how to search on data encrypted by a public-key cryptosystem. These schemes are motivated by an encrypted email system. Their constructions, however, have an overhead in search time that is proportional to the square root of the database size, which is far less efficient then the best private-key solutions. Boneh et al.'s approach [3] is known to be the seminal public key encryption scheme with keyword search (PEKS). It was observed in [2] that Boneh et al.'s scheme [3] requires a secure channel, which makes it impractical. Hence, Baek, Susilo and Safavi-Naini [2] proposed the notion of secure-channel-free PEKS to improve this drawback. This work has been further extended and revised in the recent literature, such as [12], [13]. Byun et al. [5] suggested the notion of a *keyword-guessing attack* and showed that the existing schemes are insecure against this attack, given that the number of possible keywords is bounded by some polynomial. They provided an open problem on how to construct a PEKS with designated verifier that is secure against keyword-guessing attacks. This question was answered affirmatively in [13]. In order to realize the practicality of PEKS, the combination of a public key encryption scheme with PEKS to make a single integrated entity has been studied in [1].

III. MODEL

In this section, we formalize the notion of *SPEKS* and its security requirements.

A. Syntax

SPEKS is a tuple of five algorithms, namely, ParamGen, KeyGen, IndexGen, TokenGen, Test, whose definition is given below.

- ParamGen. On input a security parameter λ, this algorithm outputs a system-wide parameter PARAM. We assume PARAM is an implicit to all the algorithms below.
- KeyGen. This algorithm outputs a secret key s.
- IndexGen. On input a keyword $w \in \{0,1\}^*$, a secret key s, this algorithm outputs a value h. The value h is called a hidden index.
- TokenGen. On input a keyword $w \in \{0,1\}^*$, a secret key s, this algorithm outputs a value ω. The value ω is called a hidden token.
- Test. On input a hidden index h, a hidden token ω, this algorithm outputs 1 or 0.

As usual, correctness is of *SPEKS* is defined as follows.

Definition 1 (Correctness): For any λ and any keyword w, $\mathsf{Test}(\text{PARAM}, h, \omega) = 1$ if there exists $\mathsf{ParamGen}(\lambda) = \text{PARAM}$, $s = \mathsf{KeyGen}(\text{PARAM})$, $h = \mathsf{IndexGen}(\text{PARAM}, s, w)$ and $\omega = \mathsf{TokenGen}(\text{PARAM}, s, w)$.

B. Typical Use of SPEKS

We briefly explain how a user Alice and the cloud server employ the algorithms in a typical scenario. Firstly, the cloud server invokes $\mathsf{ParamGen}$ to generate the parameters[1]. Alice invokes KeyGen to create her secret key s. Before outsourcing her data files F to the cloud server, Alice would choose several suitable keywords, say, w_1, ..., w_n and invokes $h_{F,i} = \mathsf{IndexGen}(s, w_i)$. She submits F along with $h_{F,i}$ to the cloud. F is possibly the encryption of Alice's data. How F is generated is out of scope of this paper.

When Alice would like to search her files from the cloud with keyword w^*, she invokes $\omega^* = \mathsf{TokenGen}(s, w^*)$ and submits ω^* to the cloud. The cloud returns all the files associated with hidden index $h_{F,i}$ such that $\mathsf{Test}(h_{F,i}, \omega^*) = 1$. Alice could share her files with keyword w^* to others by giving them the value ω^*.

C. A SPEKS System

In this section, we describe our SPEKS system. Essentially, the scheme can be divided into two phases, namely *Setup* and *Search*. We should highlight that the cloud user's storage only consists of a small and constant value, in addition to the keywords.

- **Setup:** The cloud server S runs the algorithm $\mathsf{ParamGen}$ to outputs a system-wide parameter $\textsc{param}$. The cloud user U runs the algorithm KeyGen to generate the secret key s. U then runs the algorithm $\mathsf{IndexGen}$ and algorithm $\mathsf{TokenGen}$ to generate the hidden index h and hidden token ω. Then U may now delete the original data files from his/her local storage.
- **Search:** When the cloud user U requests to search for a hidden token ω:
  1) S computes $H(\omega)$.
  2) S runs the algorithm Test, checks whether there is a hidden index h equals with $H(\omega)$, and sends U back the found data file.

  When a friend F of U requests to search for a keyword w_i:
  1) F sends the keyword w_i to U and U can returns the hidden token ω.
  2) F sends ω to S.
  3) S computes $H(\omega)$, checks whether there is a h that matches with $H(\omega)$, and returns the corresponding data file to F.

D. Security Requirements

Two security requirements are identified for *SPEKS*. The first one regards privacy. Specifically, no one, not even the cloud server, should be able to obtain information about the underlying keyword given the hidden token and hidden index. The second one concerns about the basic access control. No one should be able to compute the hidden token given the hidden index. We formalize these two requirements using security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Game Privacy.

- **Setup** Challenger $\mathcal{C}$ invokes
$$\mathsf{ParamGen}(1^\lambda) = \textsc{param}, \quad \text{and} \quad \mathsf{KeyGen}(\textsc{param}) = s.$$
$\textsc{param}$ is given to the adversary $\mathcal{A}$.
- **Query Phase 1** $\mathcal{A}$ can issue two types of queries:
  1) Index Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with
  $$\mathsf{IndexGen}(\textsc{param}, s, w).$$
  2) Token Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with
  $$\mathsf{TokenGen}(\textsc{param}, s, w).$$
- **Challenge Phase** $\mathcal{A}$ submits two keywords w_0, w_1. $\mathcal{C}$ flips a fair coin $b \in \{0, 1\}$ and computes $h_b = \mathsf{IndexGen}(\textsc{param}, s, w_b)$. h_b is returned to $\mathcal{A}$ as the challenge.
- **Query Phase 2** $\mathcal{A}$ can issue the same type of queries as in Query Phase 1 except it cannot submit queries with input w_0 and w_1.
- **Output** $\mathcal{A}$ outputs a guess bit b'. We say $\mathcal{A}$ wins the game if $b = b'$.

The advantage of $\mathcal{A}$ in Game Privacy is defined as the probability that $\mathcal{A}$ wins minus 1/2.

Game Authenticity.

- **Setup** Challenger $\mathcal{C}$ invokes
$$\mathsf{ParamGen}(1^\lambda) = \textsc{param} \quad \text{and} \quad \mathsf{KeyGen}(\textsc{param}) = s.$$
$\textsc{param}$ is given to the adversary $\mathcal{A}$.
- **Query Phase 1** $\mathcal{A}$ can issue two types of queries:
  1) Index Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with
  $$\mathsf{IndexGen}(\textsc{param}, s, w).$$
  2) Token Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with
  $$\mathsf{TokenGen}(\textsc{param}, s, w).$$
- **Challenge Phase** $\mathcal{A}$ submits one keyword w'. $\mathcal{C}$ computes $h' = \mathsf{IndexGen}(\textsc{param}, s, w')$. h' is returned to $\mathcal{A}$ as the challenge.
- **Query Phase 2** $\mathcal{A}$ can issue the same type of queries as in Query Phase 1 except it cannot submit queries with input w'.
- **Output** $\mathcal{A}$ outputs a value ω'. $\mathcal{A}$ wins the game if $\mathsf{Test}(\textsc{param}, h', \omega') = 1$.

The advantage of $\mathcal{A}$ in Game Authenticity is defined as the probability that $\mathcal{A}$ wins.

Definition 2: (Security) A construction of *SPEKS* is secure if no PPT adversary $\mathcal{A}$ can win Game Privacy or Game Authenticity with non-negligible advantage.

[1]We note that Alice could generate the parameters, but in this case, the cloud server is required to store a number of parameters since the cloud server needs to serve multiple users.

IV. Our Construction of *SPEKS*

We detail our construction of *SPEKS* in this section. We will first review the notion of hash function and pseudo-random function, which are the basic building blocks of our construction. We shall also give security analysis in after presenting our construction.

A. Building Blocks

- Hash Function. Hash functions are compressing functions that take a variable size input and return a fixed size output. Our construction relies on a one-way cryptographic hash function H. That is, given y, it is difficult to find x such that $y = H(x)$.
- Pseudo-random Function. The pseudo-random function (PRF) was first defined by [8]. It is a family of functions with the property that the input-output behavior of a random instance of the family is computationally indistinguishable from that of a random function. Informally speaking, given a random seed s, the function defined by $\textsc{Prf}(s, \cdot)$ is indistinguishable to a random function $R(\cdot)$.

B. Our Construction

- ParamGen. On input λ, output a one-way hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$.
- KeyGen. Randomly pick a pseudo-random function $\textsc{Prf} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ and a bitstring $s \in_R \{0,1\}^\lambda$ and output s.
- IndexGen. On input w and s, output $h = H(\textsc{Prf}(s, w))$.
- TokenGen. On input w and s, output $\omega = \textsc{Prf}(s, w)$.
- Test. On input a hidden index h and a hidden token ω, output 1 if and only if $h = H(\omega)$ and 0 otherwise.

C. Security Proof

Theorem 1: Our construction of *SPEKS* is secure if the pseudo-random function PRF employed is secure and the hash function H employed is one-way.

Proof: The proof is divided into two parts. In the first part, we show that if there exists an adversary $\mathcal{A}$ that has non-negligible probability in wining Game Privacy, we can construct a simulator $\mathcal{S}$ that distinguish a pseudo-random function PRF from a random function R. In the second part, we show that if there exists an adversary $\mathcal{A}$ that has non-negligible probability in wining Game Authenticity, we can construct a simulator $\mathcal{S}$ that breaks the one-way property of the hash function H.

Privacy.

- **Setup** $\mathcal{S}$ is given a function F and its goal is to distinguish if F is a random function or not. Suppose with probability 1/2 $\mathcal{S}$ is given a truly random function. $\mathcal{S}$ can query the function F adaptively. $\mathcal{S}$ chooses a one-way hash function H and set H as PARAM.
- **Query Phase 1** $\mathcal{S}$ answers the queries as follows.
  1) Index Query: $\mathcal{A}$ submits a keyword w and $\mathcal{S}$ replies with $H(F(w))$ by querying F.
  2) Token Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with $F(w)$ by querying F.
- **Challenge Phase** $\mathcal{A}$ submits two keywords w_0, w_1. $\mathcal{S}$ flips a fair coin $b \in \{0,1\}$ and computes $h_b = H(F(w))$. h_b is returned to $\mathcal{A}$
- **Query Phase 2** $\mathcal{S}$ answer $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a guess bit b'.

If $\mathcal{A}$ guess correctly, $\mathcal{S}$ concludes F is not a random function. Otherwise, $\mathcal{S}$ concluded F is a random function. Suppose $\mathcal{A}$ wins with probability $1/2+\epsilon$, probability that $\mathcal{S}$ distinguishes correctly is $1/2+\epsilon/2$. The reason is that if F is a random function, probability that $\mathcal{A}$ wins is exactly 1/2 since h_b contains no information about b. On the other hand, if F is not a random function, $\mathcal{A}$ can win with probability $1/2+\epsilon$. Thus, $\mathcal{S}$ answer correctly with probability $1/2+\epsilon/2$.

Authenticity. We use a simply game-hoping [14]. In the first game, denoted as Game Authenticity Real, the behavior of $\mathcal{S}$ is described below.

- **Setup** $\mathcal{S}$ chooses a pseudo-random function PRF with seed s and a hash function H. H is given to $\mathcal{A}$ as PARAM.
- **Query Phase 1** $\mathcal{S}$ answers the queries as follows.
  1) Index Query: $\mathcal{A}$ submits a keyword w and $\mathcal{S}$ replies with $H(\textsc{Prf}(s,\, w))$.
  2) Token Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with $\textsc{Prf}(s,\, w)$.
- **Challenge Phase** $\mathcal{A}$ submits one keywords w'. $\mathcal{S}$ returns $h_b = H(\textsc{Prf}(s,\, w'))$ to $\mathcal{A}$.
- **Query Phase 2** $\mathcal{S}$ answer $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a guess bit b'.

In the second game, Game Authenticity Modified, $\mathcal{S}$'s behavior is defined as follows.

- **Setup** $\mathcal{S}$ is given a hash function H, a value y and its goal is compute a value x such that $y = H(x)$. $\mathcal{S}$ gives $\mathcal{A}$ H as PARAM.
- **Query Phase 1** For every keyword w submitted by $\mathcal{A}$, $\mathcal{S}$ chooses a random value r_w and maintains a list of tuples (w, r_w). $\mathcal{S}$ then answers the queries as follows.
  1) Index Query: $\mathcal{A}$ submits a keyword w and $\mathcal{S}$ replies with $H(r_w)$
  2) Token Query: $\mathcal{A}$ submits a keyword w and $\mathcal{C}$ replies with r_w
- **Challenge Phase** $\mathcal{A}$ submits one keyword w'. $\mathcal{S}$ returns y to $\mathcal{A}$.
- **Query Phase 2** $\mathcal{S}$ answer $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a value ω'.

$\mathcal{A}$ wins if and only if $H(\omega') = y$. Thus, $\mathcal{S}$ outputs $x = \omega'$ as the pre-image of y. It remains to argue the advantage of $\mathcal{A}$ in Game Authenticity Real and Game Authenticity Modified are the same. A simple argument will do. If the advantage of $\mathcal{A}$ in both games are difference, it is straight-forward to use $\mathcal{A}$ to distinguishes PRF from a truly random function.

■

V. PERFORMANCE

Efficiency It is straightforward to see our construction of *SPEKS* is very efficient. Generation of a hidden index requires evaluation of one hash function and one pseudo-random function. Generation of a hidden token requires evaluation of one pesudo-random function. Testing if a hidden token matches with a hidden index requires evaluation of one hash function. Since both hash function and pseudorandom function can be implemented efficiently by heuristic algorithms, all operations of our *SPEKS* can be conducted efficiently. Indeed, they are computable even by low power hand-held devices. As for the storage, the data owner will be required to store one secret seed s. Thus, our system is very efficient and practical.

VI. CONCLUSION

We constructed an efficient *SPEKS*, which is suitable for keyword search in the cloud environment. Comparing with the existing keyword search schemes such as [11], [16], our construction is much more efficient on both sides of the data owner and cloud servers. In addition, our scheme comes with a security guarantee in the standard model.

REFERENCES

[1] J. Baek, R. Safavi-Naini, and W. Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *ISC*, pages 217–232, 2006.

[2] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *ICCSA (1)*, pages 1249–1259, 2008.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[4] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. S. III. Public key encryption that allows pir queries. In *CRYPTO*, pages 50–67, 2007.

[5] J. W. Byun, H. S. Rhee, H. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. *SDM 2006, LNCS 4165*, 75-83.

[6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[7] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[8] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[9] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[10] E. jin Goh. Secure indexes. *Technical Report 2003/216*, 2003.

[11] M. Li, S. Yu, N. Cao, and W. Lou. Authorized private keyword search over encrypted data in cloud computing. In *ICDCS*, pages 383–392, 2011.

[12] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee. Improved searchable public key encryption with designated tester. In *ASIACCS*, pages 376–379, 2009.

[13] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee. Trapdoor security in a searchable public-key encryption scheme with a designated tester. *Journal of Systems and Software*, 83(5):763–771, 2010.

[14] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[15] 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.

[16] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *ICDCS*, pages 253–262, 2010.