ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, Vol 8(16), DOI: 10.17485/ijst/2015/v8i16/65812, July 2015 ISSN (Online) : 0974-5645

A Novel Methodology to Mitigate Keyword Guessing Attack using Keyword and Signature Hash

A. Gangaa*, Nivethitha Somu and V. S. Shankar Sriram

School of Computing, SASTRA University, Thanjavur - 613401, Tamil Nadu, India; er.gangaacse@gmail.com, nivethithasomu@gmail.com, sriram@it.sastra.edu

Abstract

The importance of electronic data and its rapid growth leads to gradual increase to the data usage over the network servers and on the other side high priority is needed for the data security and privacy. The existing keyword search schemes for data retrieval from the centralized storage uses keyword as a valid parameter. Even though, keywords are encrypted and then used, intruders are able to guess keywords and can verify it using offline dictionary attack. The ability of guessing keywords by intruders reveals the stored data which affects data security and privacy. We need an additional parameter to the keyword used for search which will make the keyword guessing attack impossible. The additional parameter we used in our encryption framework is a signature file whose content needs to be independent of data. The content of signature file can be any data such as image, table, text, etc. which depends on the choice of user. This signature is hashed with keyword which makes the adversary difficult to guess the content of signature of the user. The existing approaches used keyword along with private key as search parameters, the guessing of appropriate keyword reveals the private key of user. In our framework, addition of signature along with keyword ensures both authentication as well as data privacy.

Keywords: Data Privacy, Data Retrieval, Keyword Guessing Attack, Keyword Privacy, Keyword Search Scheme, Signature Hashing

1. Introduction

Today we live in an information era, where we consume and produce massive amount of data that ranges from a simple query to large scientific computations. Table 1 shows the data generated for every minute by various social applications. Can you think of the storage capacity required to store the data generated in a day? Advancement in the storage and semiconductors technology provides us with a variety of options for data storage and processing. Most of the data generated contains personal information such as address, passwords, family photos, credit card numbers, etc. Such sensitive data needs to be stored in such a way that only authorized user can access to it. To make this feasible data security can be provided through secure authentication and encryption mechanisms which restricts the unauthorized access by insiders and outsiders.

Table 1. Data generation by various applications

Application	Data
Apple	47,000(Downloads)
Facebook	68,4478(Shares), 34,722(Likes)
Google	2000,000(Queries)
Gmail	204,166,667(Messages)
Instagram	3600(Photos)
Tumblr blog	27,778(Posts)
Twitter	100,000(Tweets)
WordPress blog	347(Posts)
YouTube	48 hours (Video)

* Author for correspondence

Consider a user a and a file server T where the user files are stored in a secure manner. To provide security, files are encrypted using any encryption algorithms and stored in the file server. Thus a user considers his files and data to be secure. What happens when it is transferred from the user or client applications to the server? Is the channel secure? If so can it prevent the attacker from accessing or replaying the content transferred? Lot of questions arises from the researches community when it covers to security. As the file server contains/stores N number of files which belongs to N number of users i.e. one user can store more than one file there needs to be a file search mechanism. In general, whenever a user needs to access his file, he searches using keyword search i.e., if a user want to access his file named cloud. datatwin. docx, he can use cloud/data/twin or combination of these keywords to search his file. Existing keyword search algorithms uses trapdoor function with search keyword and user's private key as its parameters for user validation. The problem with respect to security is that when an attacker comes to know about any one of the trapdoor function parameters, then with minimal computational power he can find another parameter.

Our proposed work on searchable encryption technique uses encryption and hashing to store and retrieve the files in the server. Files are encrypted using AES with 256-bit key, whereas using SHA-512 the hash values for the search keyword and signature are generated independently in client side. Here, search keyword is a part of the file name and the signature can be anything irrelevant to the file content. Best part of the proposed technique is that even if the attacker makes his guess on the search keyword, it is infeasible for him to guess the signature file as it nowhere related to the file accessed. It makes the keyword guessing attack harder to perform with the present computational techniques.

2. Existing Searchable Encryption Framework

Existing searchable encryption frameworks such as PEKS[1,3,4–6], etc. were based on bilinear pairing[2] and trapdoor functions. The User Authentication process involved in data retrieval from centralized storage is discussed clearly in existing frameworks[10–16] Figure 1 depicts a generic system model for data storage and retrieval in a remote server. Consider a scenario where the user wants to upload his files to a remote server. Initially user and server agree on a set of cryptographic parameters for secure file storage and retrieval. In order to store a file in a secure manner, user encrypts the file along with its associated keyword using his private key.

$$I = E_K (F, W) \quad (1)$$

Where,

I – Index of the encrypted file and keyword

K – Encryption Key (User's Public or Private Key)

F – File that needs to be stored in a secure manner on remote server

W (W_1, W_2… W_n) – Keywords related to the file name and content

Index I is created by the encryption of file and keyword using the user's private key. In order to search data the user generates Trapdoor (K, W). This trapdoor is used by the server to verify whether the given keyword is present in the index I. If it exists, then server returns the appropriate document related to that keyword.

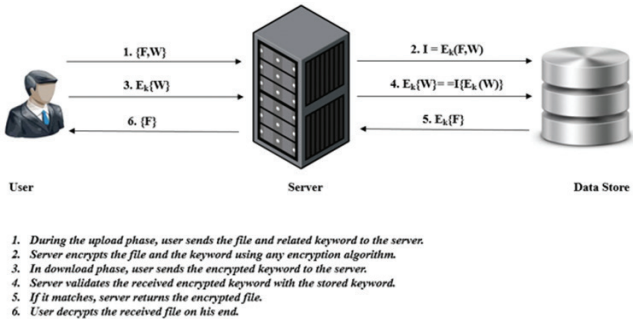

Figure 1. Generic system model for searchable encryption technique.

Whenever a file needs to be retrieved, user sends a request to the server via trapdoor function. As the trapdoor function contains the keyword and user's private key, the server validates it against the stored index. Once the validation is successful, the requested file is sent to the user, else the access to the requested file gets denied. The major drawback in the existing searchable encryption technique is that an attacker can easily perform cryptanalysis to extract the keyword or the private key from the trapdoor function.

Until now, there exist only mathematical models and proofs to ensure the security provided by the prevailing techniques. Random oracle model have been used as a benchmark to analyze the strength of existing searchable encryption techniques[7,8]. It provides restriction on the ability of an attacker to perform cryptanalysis. Hence, this model cannot ensure the level of security provided by the searchable encryption techniques in a real time scenario.

3. Proposed Secure Keyword Encryption Framework

Access to valid trapdoor function and guess on appropriate keywords makes keyword guessing attacks easy and conceivable. Based on the state of art, standard security model provides a real time analysis on the security provided by any encryption algorithm. We propose an efficient and a secure keyword search encryption framework using encryption and hashing of file and signature respectively which makes it hard for an attacker to perform keyword guessing attack.

Figure 2 depicts the proposed encryption framework which follows a simple client server system model. Whenever user wants to store his data on a remote storage he has to perform client side encryption and hashing to ensure confidentiality and integrity. Initially, the file that needs to be stored is encrypted using AES with 256 bit key. In order to provide authentication, the user creates a signature file which consists of text, images, etc. supported by any word processor.

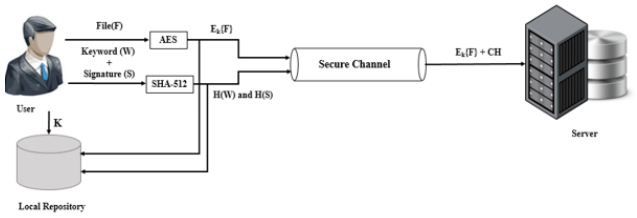

Figure 2. Proposed Searchable Encryption Techniques using AES and SHA-512.

Contents of the signature file can be related or unrelated to the file name and content. Using SHA 512, signature file and appropriate keywords are hashed to produce independent message digest values. Now, the message digest for the keyword and the signature file will be stored in the user's local repository. On the server, the encrypted file, combined hash of the keyword and the signature file are stored in a particular index I. Data transmission is made through Secure Socket Layer (SSL) which prevents the data access by the intruders in transit. Algorithm 1 explains the procedure to upload data in a remote server using AES and SHA-512.

Input - D: Data; W: Keyword; S: Signature; **Output** - E(D): Encrypted File; H(W): Hashed Keyword; H(S): Hashed Signature; CH_s: H(W+S): Hashed Keyword and Signature; **Procedure** -
1. Get **D, W** and **S** as an input from the user; 2. Encrypt **D** using **AES** with **256 bit key;** 3. Store key **K** in the local repository; 4. Hash **W** and **S** using **SHA-512;** 5. Store key **E(D), H(W)** and **H(S)** in the local repository; 6. Connect to the server using secure connection (SSH); 7. Send **E(D), H(W)** and **H(S)** to the server; 8. Compute the combined hash **(CH_s);** 9. Store the **E(D)** along with its **CH_s** in server;

Algorithm 1 – Secure File Upload to Remote Server

For data retrieval, the user submits H (W) and H(S) to the server. Once the server receives the user request, it computes the combined hash of H (W) and H(S) and compares the resulting hash value with the stored combined message digest. If a match is found, the requested document in an encrypted form is returned to the user, else the access is denied. As the user knows the key used to encrypt the file, he can easily decrypt the file. The encryption key is stored in the client side local repository. Algorithm 2 explains the procedure to download the requested file from the remote server in a secure manner.

Input - H(W); H(S); K: Key; CH_c: Computed combined hash value for validation
Output: E(D);
Procedure -

1. User connects to the server through secure connection (SSH);
2. User submits **H(W)** and **H(S)** of the requested file to the server;
3. Combined Hash (CH_c) is computed from **H(W)** and **H(S)**;
4. Compare CH_c with CH_s;
5. If (CH_c = = CH_s)
6. Return **E(D)**;
7. Disconnect the server;
8. Get **K** from the local repository;
9. Decrypt **E(D)** using **K**;
10. Else
11. Goto **Step 2**;

Algorithm 2 – Secure File Download from the Remote Server

The best part of the proposed searchable encryption framework is that even if the attacker tries to guess the keyword using Brute force attack, he fails to make a guess on the signature file as it is nowhere related to the file contents. The combined hash of signature and keyword provides a gateway to access the file. Table 2 summarizes the prevailing attacks on AES. Differential and linear cryptanalysis on SHA[9] inferred that it would require huge known and chosen plaintexts, making it hard for an attacker to decrypt SHA-512 with the current computational power which is depicted in Table 3.

Table 2. Summary of attacks on AES

Attacks	Key Size	No of Rounds	Data Com-plexity	Time Com-plexity	Memory
Square	All	6	2^{32}	2^{72}	2^{32}
Partial Sum	All	6	6×2^{32}	2^{44}	2^{32}
Collision	256	7	2^{32}	2^{192}	2^{32}
Impossible Differential	128	5	$2^{29.5}$	2^{31}	2^{42}
Boomerang	128	5	2^{39}	2^{39}	2^{33}
Impossible related-key Differential	192	7	2^{111}	2^{116}	-

4. Performance Evaluation

Consider a scenario where a user wants to store a file (word document) named guess.doc in a remote server. To ensure confidentiality and integrity, the file needs to be encrypted using the user's private key. Encrypted file does not assure complete security; hence we need to provide something on the top of it. In the proposed searchable encryption technique, we use the hash of signature and keyword for user validation. Store the keyword and signature in a separate text file guesskey.txt and guesssign.txt respectively. File encryption is carried using AES 256 bit key and the encrypted file is stored in the local repository. To create a hash of the keyword and signature file, pass the guesskey.txt and guesssign.txt as an input to the SHA-512 algorithm. Hash file named guesskey.txt.hash and guesssign.txt.hash are generated and stored in the local repository along with the encrypted file. User uploads the encrypted file, guesskey.txt.hash and guesssign.txt.hash to the server, where the combined hash is generated using the given hash of keyword and signature. Encrypted file along with the combined hash is stored on the server in an index.

Table 3. Comparisons on SHA based on Key size

Algorithm	Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds
SHA2-224	224	256	512	2^{64}-1	64
SHA2-256	256	256	512	2^{64}-1	64
SHA2-384	384	512	1024	2^{128}-1	80
SHA2-512	512	512	1024	2^{128}-1	80
SHA3-224	224	1600	1152	Unlimited	24
SHA3-256	256	1600	1088	Unlimited	24
SHA3-384	384	1600	832	Unlimited	24
SHA3-512	512	1600	576	Unlimited	24

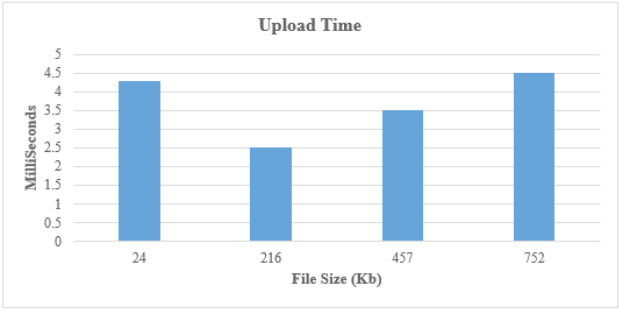

Figure 3. Time taken to upload file of different sizes.

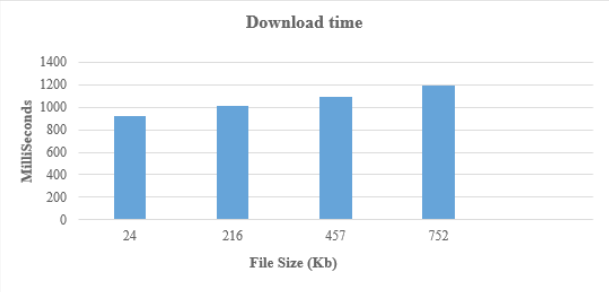

Figure 4. Time taken to download file of different sizes.

To retrieve the file from the server user submits guesskey.txt.hash and guesssign.txt.hash. Server computes the combined hash of the given input and compares the resulting hash with the stored hash value. If a match exists, then the server returns the encrypted file requested by the user. On the client side as the user has his key, decrypts the file. Existing searchable encryption techniques performs server side processing for encryption and hashing. In our proposed work, we perform the entire computation on the client side and store the encrypted file and combined hash value of W+K. We have also used signature in addition with the keyword and hashed them to enhance confidentiality and integrity for making it hard for the attacker to perform any type of guessing attack.

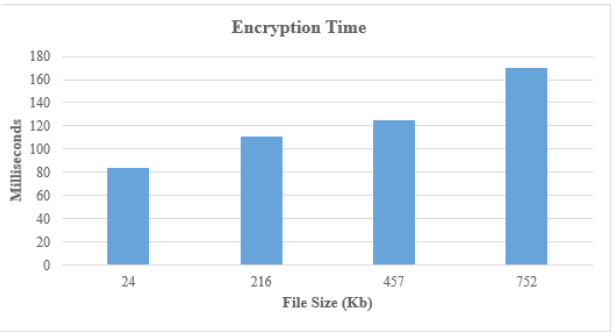

Figure 5. Time taken to encrypt file of different sizes.

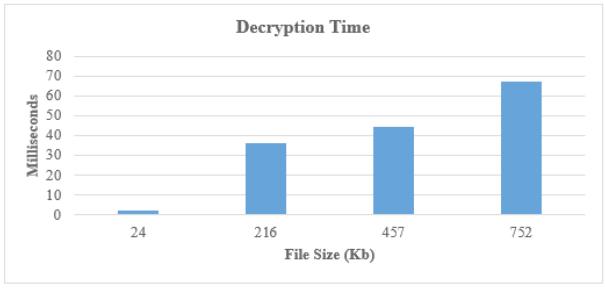

Figure 6. Time taken to decrypt file of different sizes.

Figure 3, 4, 5 and 6 illustrates the time taken to upload, download, encrypt and decrypt files of various sizes. From this, we can conclude that with a minimal configured system client side can process all the encryption and hashing function. Hence, reduces the communication overhead and transfer of sensitive data between the client and the remote server.

5. Conclusion

We proposed a secure encryption framework, wherein the files are encrypted using the user's private key where the hash of the signature plus keyword are provided to ensure data confidentiality and integrity. The best part of this technique is that even when the attacker can make a guess on the keyword it is hard for him to find the signature as it is nowhere related to the stored file. Experimental results shows that with the computing resource provided by the client system, we can process files of various sizes thus reducing the communication overhead and the transfer of sensitive data between two hosts.

6. References

1. Boneh D, Di Crescenzo G, Ostrovsky R, Persiano G, editors. Public key encryption with keyword search. Advances in Cryptology-Eurocrypt. 2004; 3027:506–22.
2. Waters BR, Balfanz D, Durfee G, Smetters DK, editors. Building an Encrypted and Searchable Audit Log. 11th Annual Network and Distributed Security Symposium (NDSS '04); 2004. p.1.
3. Fang L, Susilo W, Ge C, Wang J. A secure channel free public key encryption with keyword search scheme without random oracle. Cryptology and Network Security. 2009; 5888:248–58.
4. Park DJ, Kim K, Lee PJ. Public key encryption with conjunctive field keyword search. Information Security Applications. 2005; 3325:73–86.
5. Fang L, Susilo W, Ge C, Wang J. Public key encryption with keyword search secure against keyword guessing attacks without random oracle. Information Sciences. 2013; 238:221–41.
6. Bellare M, Rogaway P, editors. Random oracles are practical: A paradigm for designing efficient protocols. Proceedings of the 1st ACM conference on Computer and communications security; 1993; ACM; p. 62–73.
7. Canetti R, Goldreich O, Halevi S. The random oracle methodology, revisited. Journal of the ACM (JACM). 2004; 51(4):557–94.
8. Biham E, Keller N, editors. Cryptanalysis of reduced variants of Rijndael. 3rd AES Conference; 2000; New York, USA; p. 1–11.

9. Ganeshkumar K, Arivazhagan D. Generating a digital signature based on new cryptographic scheme for user authentication and security. Indian Journal of Science and Technology. 2014 Oct; 7(S6):1–5.
10. Mohit K, Ghrera SP. A new group key transfer protocol using CBU hash function. Indian Journal of Science and Technology. 2014 Jan; 7(1):19–24.
11. Lee J-Y. A Study on the use of secure data in cloud storage for collaboration. Indian Journal of Science and Technology. 2015 Mar; 8(S5):33–6.
12. Gomathi K, Parvathavarthini B. An enhanced distributed weighted clustering routing protocol for key management. Indian Journal of Science and Technology. 2015 Feb; 8(4):342–8.
13. Boopathi K, Sreejith S, Bithin A. Learning Cyber Security through Gamification. Indian Journal of Science and Technology. 2015 Apr; 8(7):642–9.
14. Mritha R, Isa NAM. A steganography approach over video images to improve security. Indian Journal of Science and Technology. 2015 Jan; 8(1):79–86.
15. Ganeshkumar K, Arivazhagan D. Generating a digital signature based on new cryptographic scheme for user authentication and security. Indian Journal of Science and Technology. 2014 Oct; 7(S6):1–5.
16. Nishioka M. Perfect keyword privacy in PEKS systems. Provable Security. 2012; 7496:175–92.