Implementing the Honey Encryption for Securing Public Cloud Data Storage

Edwin Mok[1], Azman Samsudin[2], Soo-Fun Tan[3]
{edwinmyp@gmail.com[1], azman.samsudin@usm.edu.my[2], soofun4818@yahoo.com[3] }

School of Computer Sciences, Unversiti Sains Malaysia, Penang, 11800, Malaysia

Abstract. Recent security incidents on public cloud data storage had risen concerns on cloud data security. Existing cloud data protection solutions that primarily relying on the conventional password-based encryption cannot efficiently resist password guessing and password cracking attacks. To address this problem, this paper proposed an eXtended Honey Encryption (XHE) scheme by adding an additional protection mechanism on the encrypted data. When the attacker attempts to access these encrypted data by entering the incorrect password, instead of rejecting the access, the HE algorithm generates an indistinguishable bogus data, in which the attack could not determine whether the guessed password is working correctly or not. Therefore, increasing the complexity of password guessing and cracking attacks.

Keywords: Cloud, Data Centric Approach, Password-based Encryption, Honey Encryption.

1 Introduction

Traditionally, the data encryption algorithms are considered as “computationally secure", if the best known method of breaking the algorithms require an unreasonably large amount of computer processing time [1]. However, with the advancement of computing processors, parallelism techniques and distributed algorithms, existing cloud data protection that relies on the conventional password-based data encryption algorithms (e.g. Advanced Encryption Standard (AES) [2,3], RSA [4,5], etc.) are constantly at risk of being challenged and broken [6]. For instance, the recent attack reported in [7] can brute force any eight-characters-length password that consists combination of any 95 characters in less than 5.5 hours.

To address this problem, Juels and Ristenpart [8,9] proposed a Honey Encryption (HE) scheme to enhance the security of a password-based encrypted data. When the attacker attempts to access the encrypted data with the incorrect password, instead of rejecting their data access, the HE algorithm generates an indistinguishable bogus data that closely resemblance the actual data. Subsequently, the attacker will be bewildered in determining whether the guessed password is working correctly or not. Therefore, increasing the complexity of password guessing and cracking process. This paper extends the HE scheme to enhance the security of the cloud data storage, so called as eXtended Honey Encryption (XHE) scheme. The rest of this paper is organized as follows. Section 2 describes preliminaries and some background works of

HE scheme. Section 3 presents the algorithm of XHE scheme. Section 4 demonstrates the applicability of XHE scheme to enhance the data security on public cloud service providers. Finally, this paper draws a conclusion in Section 5.

2 Preliminaries

Honey Encryption (HE) scheme [8,9] was firstly introduced by Juels and Ristenpart on 2014 to add the extra protection layer onto the password-based RSA encryption algorithm and credit card applications. Subsequently, extended by Tyagi et al. [10] and Huang et al. [11] to secure the basic text messaging and genomic data application respectively. More recently, Joseph et al. [12] enhanced the security of HE scheme to resist the message recovery attacks. This section introduces some concepts and background which will be used in the construction of the extended HE scheme in Section 3.

Message Space *($\mathcal{M}$)* [8,9]. Since HE deceives the attackers by providing ambiguous looking messages, it requires a message space, $\mathcal{M}$, which contains all possible messages, M. The size of $\mathcal{M}$ has to be customized for each scenario and dependent on the type of contents that need to be encrypted. The distribution over $\mathcal{M}$ is denoted as ψ_m, subsequently, sampling according to this distribution is denoted as $M \leftarrow \psi_m \mathcal{M}$.

Seed Space *(S)* [8,9]. Seed space, S, is the space of all n-bit binary strings for some predetermined n. Each message in $\mathcal{M}$ is mapped to a seed in S. The size of the seed is directly proportional to how likely a particular message is to appear. The size also has to be large enough at such even the least likely messages have to be mapped to at least 1 seed. Similar to $\mathcal{M}$, S is predefined by developer which can be based on personal judgment, research or sampling results. The distribution on set S is denoted as a map $p: S \rightarrow [0,1]$ such that $\Sigma_{s \in S} p(s) = 1$. Subsequently, sampling according to such distribution is denoted as $s \leftarrow_p S$.

Distribution-Transforming Encoder (*DTE*) [8,9]. A *DTE* consists of a pair of algorithms, such that $DTE = (encode, decode)$. The *encode* algorithm takes as input a message $M \in \mathcal{M}$ and outputs a set of seed value, S from seed space, $\mathcal{S}$. The deterministic *decode* algorithm takes as input a message $S \in \mathcal{S}$ and outputs a message $M \in \mathcal{M}$. The correctness of DTE algorithm follows as for any $M \in \mathcal{M}$, $Pr[decode (encode (M)) = M] = 1$.

Inverse Sampling DTE (*IS-DTE*) [8,9]. A *IS-DTE* consists of a pair of algorithms, such that IS-DTE = $(IS\text{-}encode, IS\text{-}decode)$. The *IS-encode* algorithm runs the Cumulative Distribution Function (CDF), F_m such that with a pre-defined message distribution ψ_m and $\mathcal{M} = \{M_1, M_2, ..., M_{|\mathcal{M}|}\}$. Define $F_m(M_0) = 0$, subsequently generates M_i such that $F_m(M_{i-1}) \leq S < F_m(M_i)$, where $S \leftarrow_\$ [0,1)$. Lastly, encodes the input message M_i by selecting a uniformly random value from the range $[F_m(M_{i-1}), F_m(M_i)]$. The *IS-decode* algorithm is the inverse of CDF, such that $IS\text{-}decode = F_m^{-1}(S)$.

DTE-then-Encrypt (*HE [DTE, SE]*) [8,9]. A HE [DTE, SE] algorithm is a pair of algorithms (*HEnc, HDec*) that encrypts a message by using the *DTE* algorithm, subsequently re-encrypts the output of *DTE* algorithm with Symmetric Encryption scheme (*SE*) as follows.

HEnc (K, M). Given the symmetric key, K and a message M, let the H be the hashing algorithm and n is the number of random bits, select a uniformly random, $s \leftarrow_{\$} encode(M)$ and $R \leftarrow_{\$} \{0,1\}^n$, outputs the ciphertext, $C = H(R,K) \oplus s$. The process of HEnc ***(K, M)*** is illustrated in Fig.1.

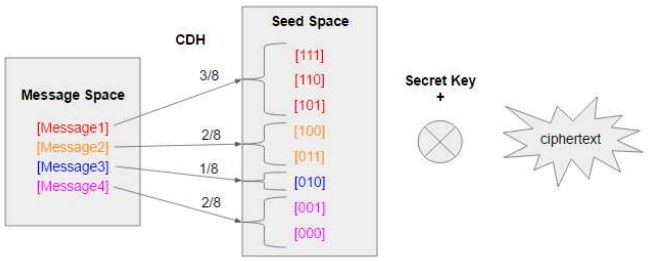

Fig. 1. The process of *HEnc* algorithm.

HDec (K, R, C). Given the K， R and C, computes $s = C \oplus H(R, K)$ and subsequently outputs the ciphertext, $M= decode(s)$. The seed, s, alone is insufficient to retrieve the message, M, unless it is a one-to-one mapping. In most cases, a s falls into a seed range, S. Therefore, Inverse sampling table comes into play for message lookup as illustrated in Fig. 2.

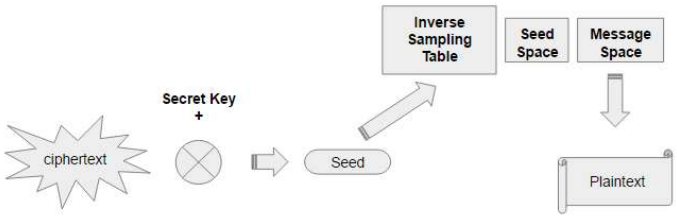

Fig. 2. The process of *HDec* algorithm.

3 EXtended Honey Encryption (XHE) Scheme

To be adapted for securing the public cloud file storage, this paper proposes an extended version of the HE scheme [8,9], so called as eXtended Honey Encryption (XHE) scheme. Similar to credit card applications, the length of the file name usually has a limit size. For instance, Linux Ext4 file system has a maximum of 255 characters per file and most of the time, these file and folder names do not exceed more than 50 characters [13]. Subsequently, the file extension such as .exe, .bat, .sh, .txt, .docx, .pdf, .jpeg, etc., can be randomly assigned to a file to increase the complexity against the password attacks. To achieve this, this paper proposed XHE algorithm and subsequently divided it into 2-sub algorithms in order to protect the file's name and file's extension respectively. Next, the construction of the extended HE scheme is presented in the following.

Message Space $(\mathcal{M}_1, \mathcal{M}_2)$. Given the message, M_1 is a file name that consists of 36 alphanumeric characters with the maximum length of 50 characters [13-15]. With the total of 36^{50} possibilities, the distribution over $\mathcal{M}_1$ is denoted as ψ_{m1} and the sampling according to such distribution is denoted as $M_1 \leftarrow \psi_{m1} \mathcal{M}_1$, as illustrated in Fig. 3.

Fig. 3. The Message space, M_1 of the Extended HE scheme.

Next, let M_2 be the file extension that consists of alphabet with the maximum length of 4 characters [16] as illustrated in Fig. 4. With a list of most common used file extensions [16], the distribution over $\mathcal{M}_2$ is denoted as ψ_{m2} and the sampling according to such distribution is denoted as $M_2 \leftarrow \psi_{m2} \mathcal{M}_2$.

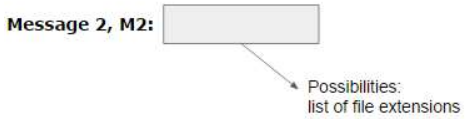

Fig. 4. The Message space, M_2 of the Extended HE scheme.

Seed Space (S_1, S_2). Seed space, consists of prefix seed, S_1, and suffix seed, S_2, over the n-bit binary strings. Each message in $\mathcal{M}_1$ and $\mathcal{M}_2$ are mapped to a seed in S_1 and S_2 respectively such that $\Sigma_{s1 \in S1} p(s_1) = 1$ and $\Sigma_{s2 \in S2} p(s_2) = 1$.

Distribution-Transforming Encoder (*DTE*). A *DTE* consists of a pair of algorithms, *DTE_1* and *DTE_2* algorithms as follows:

DTE_1(encode_1, decode_1). The *encode_1* algorithm takes a file name as input, $M_1 \in \mathcal{M}_1$ and outputs a set of prefix seed value, s_1 from seed space, S_1. The deterministic *decode_1* algorithm takes as input a message $s_1 \in S_2$ and outputs a message $M_1 \in \mathcal{M}_1$.

DTE_2(encode_1, decode_2). The *encode_2* algorithm takes a file extension as input, $M_2 \in \mathcal{M}_2$ and outputs a set of suffix seed value, s_2 from seed space, S_2. The deterministic *decode_2* algorithm takes as input a message $s_2 \in S_2$ and outputs a message $M_2 \in \mathcal{M}_2$ by running the binary search on the inverse sampling table and linear search on the message space to locate the original file name and extension type.

Fig. 5. The *DTE* algorithm $_{of}$ the Extended HE scheme.

Inverse Sampling DTE (*IS-DTE*). A IS-*DTE* consists of a pair of algorithms, *IS-DTE_1* and *IS-DTE_2* algorithms as follows:

IS-DTE_1(IS-encode_1, IS-decode_1). The *IS-encode_1* algorithm runs the Cumulative Distribution Function (CDF), F_{m1}, such that with a pre-defined message distribution, ψ_{m1} and $\mathcal{M}_1 = \{M_{1\text{-}1}, M_{1\text{-}2}, \ldots, M_{1\text{-}|\mathcal{M}|}\}$. Define $F_{m1}(M_{1\text{-}0}) = 0$, subsequently generates $M_{1\text{-}i}$ such that $F_{m1}(M_{i\text{-}1}) \leq S_1 < F_{m1}(M_{1\text{-}i})$, where $S_1 \leftarrow_{\$} [0,1)$. Lastly, encodes the input message $M_{1\text{-}i}$ by selecting a uniformly random value from the range $[F_{m1}(M_{(1\text{-}i)\text{-}1}), F_{m1}(M_{1\text{-}i})]$. The *IS-decode_1* algorithm is the inverse of CDF, such that *IS-decode_1* $= F_{m1}^{-1}(S_1)$.

IS-DTE_2(IS-encode_2, IS-decode_2). The *IS-encode_2* algorithm runs the Cumulative Distribution Function (CDF), F_{m2}, such that with a pre-defined message distribution, ψ_{m2} and $\mathcal{M}_2 = \{M_{2\text{-}1}, M_{2\text{-}2}, \ldots, M_{2\text{-}|\mathcal{M}|}\}$. Define $F_{m2}(M_{2\text{-}0}) = 0$, subsequently generates $M_{1\text{-}i}$ such that $F_{m2}(M_{i\text{-}1}) \leq S_2 < F_{m2}(M_{2\text{-}i})$, where $S_2 \leftarrow_{\$} [0,1)$. Lastly, encodes the input message $M_{2\text{-i}}$ by selecting a uniformly random value from the range $[F_{m2}(M_{(2\text{-}i)\text{-}1}), F_{m2}(M_{2\text{-}i})]$. The *IS-decode_2* algorithm is the inverse of CDF, such that *IS-decode_2* $= F_{m2}^{-1}(S_2)$.

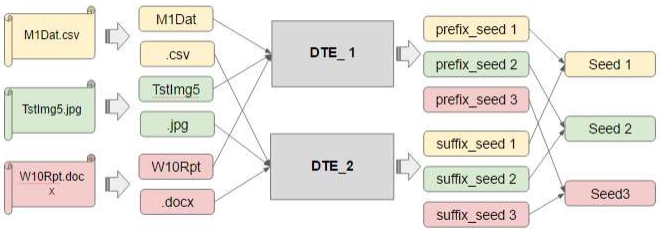

Fig. 6. The *DTE* and *IS-DTE* encoding algorithm of the Extended HE scheme.

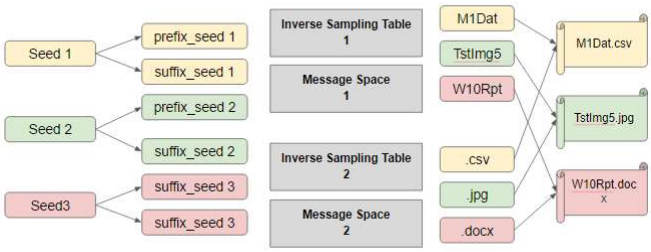

Fig. 7. The *DTE* and *IS-DTE* decoding of the Extended HE scheme.

DTE-then-Encrypt (*HE [DTE, SE]*). A HE [DTE, SE] algorithm is a pair of algorithms (*HEnc, HDec*) that encrypts a message by using the *DTE* algorithm, subsequently re-encrypts the output of *DTE* algorithm with a Symmetric Encryption scheme (*SE*) as follows.

HEnc (K, M_1, M_2). Given the symmetric key *K,* and a file name M_1, and its extension M_2, let the *H* be the hashing algorithm and *n* is the number of random bits, select a uniformly random, $s_1 \leftarrow_{\$} encode(M_1)$, $s_2 \leftarrow_{\$} encode(M_2)$ *and* $R \leftarrow_{\$} \{0,1\}^n$, outputs the ciphertext, $C_1 = H(R,K) \oplus s_1$ and $C_2 = H(R,K) \oplus s_2$.

HDec (K, R, C_1, C_2). Given the *K*, *R*, C_2 and C_2 computes $s_1 = C_1 \oplus H(R, K)$ and $s_2 = C_2 \oplus H(R, K)$. Subsequently outputs the file name, $M_1 = decode(s_1)$ and its extension, $M_2 = decode(s_2)$ with the lookup inverse sampling tables.

4 Extended Honey Encryption in Securing Cloud Data Storage

This section describes the applicability of the XHE scheme to enhance the security of file storage on public cloud environment as illustrated in Fig. 8. The application scenario is further described in the following.

System Setup(λ). Given a security parameter, λ, define the distribution over the key space, ψ_k. Subsequently, takes the user's cloud account login password as an input, and outputs the shared secret key, *K*.

Encrypt (*K, F*). Given the secret key, *K* and a file, *f* (e.g. word documents, images, etc.), takes a *f* as an input, extract the file name, as a message space M_1, and file extension as a message space, M_2. Subsequently, define the distribution over the messages space ψ_{m1} and ψ_{m2}. Next, generates the seed s_1 with *DTE_1 encode_1* algorithm and s_2 with *DTE_2 encode_2* algorithm. Then, runs the *IS-encode_1* algorithm to generate a series of fake file names and its contents that looks like an original file M_1, such that $F_{1\text{-}i} = \{F_{m1}(M_{(1\text{-}i)\text{-}1}), F_{m1}(M_{1\text{-}i})$. Subsequently, executes the *IS-encode_2* algorithm to generate a series of fake file extensions that close related to an original file extension M_2, such that $F_{2\text{-}i} = \{F_{m2}(M_{(2\text{-}i)\text{-}1}), F_{m2}(M_{2\text{-}i})\}$. Lastly, outputs the encrypted file, $f_e = C_1.C_2$ with *HEnc (K, M_1, M_2)* algorithm.

Data Transmission and Storage. The user uploads his encrypted files, f_e, onto public cloud service providers' data centre. These files are transferred via the existing Transport Layer Security (TLS) and stored onto public cloud with protection.

Decrypt (K,F_e). The user downloads his encrypted file, f_e, from the public cloud service provider and runs *HDec* algorithm to recover his plain file with his secret key, K.

The toy example is described as follow. The user encrypts her file, f = *"2016.docx"* under the shared secret key, $K = 0000$. Next, the user defines the distribution over the messages space, $\psi_{m1} = \{2014, 2015, 2016, 2017\}$ and $\psi_{m2} = \{xlsx, docx, txt, ppt\}$. Next, generates the 2-bit strings seed, $S_1 = \{00, 01, 10, 11\}$ and $S_2 = \{00, 01, 10, 11\}$ with *DTE* algorithm. Then, the user encodes her file as $encode\ (2016) = s_1 = 10$ and $encode\ (docx) = s_2 = 01$. Next, the user selects the random 2-bit strings, R and outputs the ciphertext, $C_1 = H(R,K) \oplus s_1 = H(R,0000) \oplus 10 = 11 \oplus 10 = 01$ and $C_2 = H(R,0000) \oplus 00 = 11 \oplus 01 = 10$. To recover the original file, the user computes $s_1 = C_1 \oplus H(R,K) = 01 \oplus 11 = 10$ and $s_2 = C_2 \oplus H(R,K) = 10 \oplus 11 = 01$. Subsequently, outputs the $decode\ (s_1) = decode(10) = 2016$ and $decode\ (s_2) = decode(01) = docx$.

Attack Scenario. Suppose that the adversary intercepts the encrypted file, f_e, either from the user's device (e.g. desktop, tablets, mobile devices, etc.), public cloud data storage or during the data transmission, and subsequently attempts to decrypt it. With his guessed password, the *Decrypt* algorithm outputs a fake file in response to every incorrect guess of the user's password or shared symmetric key, K. These fake file is generated with the *IS-encode* algorithm in which the fake file distribution is closed related to actual file distribution. Therefore, it is indistinguishable from the attacker perspective, thus increases the complexity of determining whether the attacker have guessed a password correctly or not. For instance, the adversary uses the most popular password, $K = 1234$ and $g = H(R,1234) = 01$. The adversary computes $s_1 = C_1 \oplus g = 01 \oplus 01 = 00$ and $decode\ (00) = 2014$. Subsequently, computes $s_2 = C_2 \oplus g = 10 \oplus 01 = 11$ and $decode\ (11) = ppt$ and obtain a valid file named "*2014.ppt*".

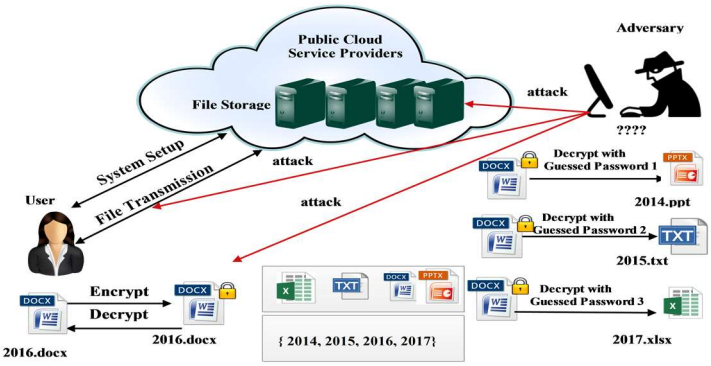

Fig. 8. Secure Cloud File Storage with the Extended Honey Encryption.

5 Conclusions

While the existing file protection relies on password-based encryption, which is vulnerable to password guessing attack such as brute-force, dictionary or rainbow table attack [6], this paper proposed an eXtended Honey Encryption (XHE) scheme for securing the public cloud file storage. The proposed XHE scheme provides an additional protection layer to existing encrypted file. When the attacker attempts to access the encrypted data with his guessing password, instead of rejecting their data access as conventional file encryption scheme, the extended HE algorithm generates an indistinguishable bogus file that are closely related to the original file. It is noticeable that the message space of the proposed scheme is pre-fixed and the complexity and the size of inverse sampling tables is growing exponentially with the increase of the file names and its extension sizes. In future, several aspects of this work can be further explored, such as working with a flexible message space and further extends into folders protection. Besides that, whether the proposed XHE scheme can be further adapted to work with the recent advancement of cryptography algorithm such as Homomorphic Encryption [17] for supporting the computation on encrypted data, as well as Attribute-Based Encryption (ABE) [18] to fine grained control access on encrypted data are another interesting topic to be explored.

Acknowledgments. This work was by a research grant from Universiti Sains Malaysia (USM) [1001/PKOMP/811334]. The authors also thank the anonymous reviewers of this manuscript for their careful reviews and valuable comment

References

[1] W. Diffie and M. E. Hellman: New Directions in Crytography. IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644 -654. IEEE Press, New Jersey. (1976).

[2] G. Irazoqui, M. S. Inci, T. Eisenbarth and B. Sunar: Wait a Minute! A Fast, Cross-VM Attacks on AES. LNCS, vol. 8688, pp. 299-319, Springer, Switzerland, 2014.

[3] Y. Wei, J. Lu and Y. Hu: Meet-in-the-Middle Attack on 8 Rounds of the AES Block Cipher under 192 Key Bits. LNCS, vol. 6672, pp. 222-232, Springer, Heidelberg. (2011)

[4] A. Nitaj, M. R. K. Ariffin, D. I. Nassar, H. M. Bahig: New Attacks on the RSA Cryptosystem. LNCS, Progress in Cryptology – AFRICACRYPT, LNCS, vol. 8469, pp. 178--198. Springer, Swtizerland. (2014).

[5] Y. Lu, L. Peng, S. Sarkar.: Cryptanalysis of an RSA variant with Moduli $N = p^r q$. In: 9th International Workshop on Coding and Cryptography 2015 WCC2015, Apr 2015, Paris, France. 2016

[6] S.F. Tan and A. Samsudin: Enhanced Security for Public Cloud Storage with Honey Encryption. Advanced Science Letters. Accepted Manuscript.

[7] J. M. Gosney: Password Cracking HPC. In: Pawword^12 Security Conference, Norway: Olso (2012).

[8] A. Juels and T. Ristenpart.: Honey Encryption: Encryption Beyond the Brute-Force Barrier. IEEE Security and Privacy, vol. 12, no.4, pp. 59--62. IEEE Press, New York. (2014)

[9] A. Juels and T. Ristenpart, Honey Encryption: Security Beyond the Brute-Force Bound. Advances in Cryptology – Eurocrypt, LNCS, vol. 8841, pp. 293--310. Springer, Heidelberg. (2014)

[10] N. Tyagi, J. Wang, K. Wen and D.Zuo.: Honey Encryption Applications. 6.857 Computer and Network Security, Massachusetts Institute of Technolog. (2015) http://www.mit.edu/~ntyagi/papers/honey-encryption-cc.pdf
[11] Z. Huang, E.Ayday, J. Fellay, J-P. Hubuax and A. Juels.: GenoGuard: Protecting Genomic Data Against Brute-Force Attacks. In: IEEE Symposium on Security and Privacy, pp. 447--462. IEEE Press, California. (2015)
[12] J. Joseph, T. Ristenpart and Q.Tang.: Honey Encryption Beyond Message Recovery Security, IACR Cryptology ePrint Archive, pp. 1—28. (2016).
[13] A. Mathur, M. Cao and S. Bhattacharya, Suparna. : The New Ext4 Filesystem: Current Status and Future Plans. In: Linux Symposium, pp 21--34. Red Hat, Ontario (2007).
[14] MSDN.: Windows Naming Conventions. Retrieved 3 May 2016. https://msdn.microsoft.com/en-us/library/aa365247.aspx#naming_conventions
[15] OS X: Cross-platform filename best practices and conventions. Retrieved 3 May 2016. https://support.apple.com/en-us/HT202808
[16] FileExt.: File Extension. Retrieved 3 May 2016. http://www.fileext.com/
[17] S.F. Tan and A. Samsudin.: A Survey of Homomorphic Encryption for Outsourced Big Data Computation, KSII Transaction on Internet and Information Systems, vol. 10, No. 8, Aug. 2016.
[18] S. F. Tan and A. Samsudin.: Lattice Ciphertext Policy Attribute based Encryption from Ring-LWE, In: 2nd International Symposium on Technology Management and Emerging Technologies (ISTMET 2015) Langkawi, Malaysia. 25-27, pp. 282-286. (2015).