2010 16th International Conference on Parallel and Distributed Systems

# BLAST*: Applying Streaming Ciphers into Outsourced Cloud Storage

Ki-Woong Park, Chulmin Kim, and Kyu Ho Park
Computer Engineering Research Laboratory
Korea Advanced Institute of Science and Technology
{woongbak, cmkim, kpark}@core.kaist.ac.kr

*Abstract*— **Providing secure and efficient outsourced storage is a precursor to widespread cloud deployment and availability. For this purpose, existing designs mainly rely on block ciphers, although stream ciphers are more computationally-efficient than block ciphers. This paper presents a construction of secure storage, BLAST, enhanced with a stream cipher rather than a block cipher with a novel block accessible encryption mechanism based on streaming ciphers. In BLAST, a hierarchical tree generated in the form of an n-level quad tree is created for each user during the registration phase of the storage system. When a user wants to access the outsourced storage, the user can access their data after encrypting or decrypting it using a sequence of key stream frames derived from the hierarchical tree. The experimental results show that the proposed system achieves significant improved performance than normal streaming/block cipher-based secure storage system in terms of throughput and access latency.**

***Keywords: Secure Storage, Storage Outsourcing, Streaming Cipher, Block Cipher***

## I. INTRODUCTION

Shifting data into cloud storage systems such as the Amazon Simple Storage Service (S3) [1] and Azure [2] offers great convenience to users, as it liberates them from concerns about the scalability and complexity of storage management. While these storage outsourcing services provide elastic storage space on demand, users hesitate to expose potentially sensitive data to cloud computing or commit such data to cloud storage. It is due to concerns about security in cloud environments where infrastructure elements such as networks and storage, may be shared among many different organizations. Despite the potential advantages, this computing platform shift poses new and challenging issues [3]. It requires both confidentiality and computation efficiency because bulky security operations should be performed on a thin client.

As a fundamental means of enforcing confidentiality, block ciphers and stream ciphers stand out as widely used mechanisms. A block cipher is a type of symmetric-key encryption algorithm that encrypts data in blocks rather than encrypting one bit at a time in a stream, which is known as a stream cipher. Among these, the stream cipher is more computationally-efficient than the block cipher, as the encryption of data using a stream cipher is simply based on the XORing of the stream with the original data in a bitwise configuration [4].

* BLAST: Block-Accessible Streaming Cipher-Based Secure Cloud Storage

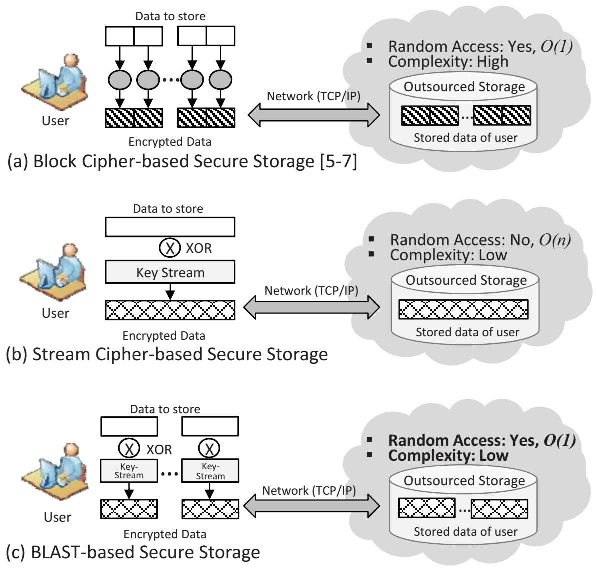
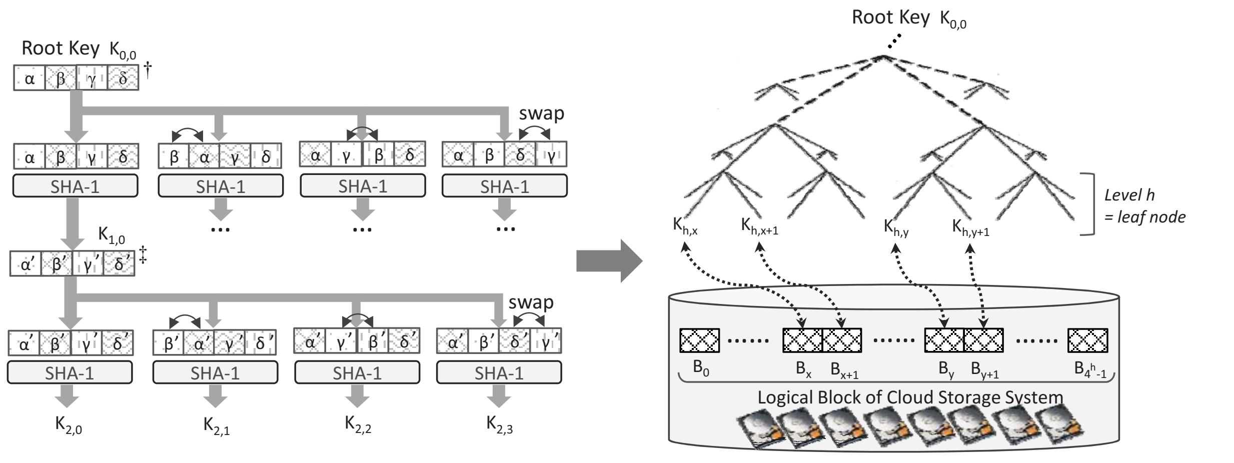


**Figure 1. Overview of Secure Storage System**

In spite of the efficiency of the stream cipher, existing secure storage system designs mainly rely on block ciphers because it is commonly believed that stream ciphers are not suitable for random access. In case of stream ciphers, to decrypt a certain block, one generally has to run the underlying key stream generator forward from the initial state of the secret key up to the internal state of the corresponding block. Figure 1 presents the proposed approach (Fig. 1c) in comparison with normal secure storage systems based on a block cipher or a stream cipher (Figs. 1a, 1b). Block cipher-based secure storage systems such as those in a recent study [5-7] can provide block accessibility with *O(1)*-time planarity but have relatively high operation complexity. The stream cipher-based secure storage system, on the other hand, can have lower cryptographic operation overhead only for sequential access patterns due to the missing block accessibility, which has *O(n)*-time planarity, where *n* is access address.

The main objective of this paper is to present a block accessible encryption mechanism with *O(1)*-time planarity based on streaming ciphers. This is termed BLAST, and enables the construction of a secure outsourced-storage system based on a stream cipher rather than on a block cipher. In BLAST, a key derivation tree, which is generated by a one-way hash function in a type of n-level quad tree, is created uniquely for each user during the registration process.



(a) Key Tree Construction by Hash and Swap

(b) Key Mapping with Logical Block of Outsourced Storage

† Key is split into four subsections (α, β, γ, δ)
‡ Hashed value (=derived key) by SHA-1, key is split into four subsections (α′, β′, γ′, δ′)

$K_{a,b}$ a: level in the tree
b: sequence number in the level

**Figure 2. Key Tree Construction and Mapping Keys with Logical Block of Outsourced Storage**

When a user wants to access outsourced storage, the user can upload/download their data after encrypting or decrypting it using a sequence of key stream frames derived from the hierarchical tree. The proposed scheme is more computationally-efficient than the normal streaming or block cipher-based secure storage system in terms of its throughput and access latency by the block accessible encryption mechanism based on streaming ciphers. There has been prior work on storage system enhanced with security in the research area of cloud computing such as [5-11]. Their work is orthogonal to ours, in the sense that our proposed scheme can be deployed to their work without changing the basis of their system. In addition, we note that our scheme can be realized using any kind of streaming ciphers such as SEAL [19], Rabbit [24], and RC4 [18] etc., which have relatively lower computation complexity than block ciphers. In our cloud platform which is a part of Peta-scale cloud computing platform development project [25, 26], BLAST has been integrated into Secure Storage Service as the key primitive for outsourced data storage over the network. It can be also extended to other application areas, including a secure content distribution network [15] and secure file sharing.

This paper is organized as follows. Sections 2 and 3 explain the proposed scheme and show the experimental results and safeness. Section 4 discusses the related work of this study. Conclusions and future work directions are presented in Section 5.

## II. BLAST Operation

While streaming ciphers are more efficient than block ciphers for data sizes larger than 100 bytes [4], streaming ciphers have not been deployed in secure storage system. The main reason is that one generally has to run the underlying key stream generator forward from the initial state of the secret key up to the internal state of the corresponding block to access. In this section, the details of the proposed approach to deploy streaming ciphers in secure storage are presented. The underlying assumptions of the proposed system are initially given and the key-derivation and access procedure mechanisms are then discussed.

### *A. Assumptions*

The underlying assumptions for the proposed scheme are defined based on other recent research [10, 11, 15] which treated the security issues of outsourced storage. It is assumed that the atomic access unit for outsourced storage is a 'block'. In the proposed scheme, encryption is performed at the defined block level to provide block accessibility. Hence the 'block' size is aligned with the block size of a file or storage system. It is assumed that a predefined shared secret key and integrity mechanisms between the user and service provider exist, which implies that the user has information to verify the integrity of the received data, such as MACs[2], as described in two recent studies [8, 9]. To preserve the confidentiality of the outsourced data, the user may ask the service provider to conduct one additional encryption process (over-encryption) [12] before data transactions occur between the user and service provider. This paper focuses on confidentiality and computation efficiency to present a block accessible encryption mechanism based on streaming ciphers to construct a secure outsourced storage system based on a stream cipher rather than on a block cipher.

### *B. Key Tree Construction*

To provide block accessibility in the secure storage system based on the stream ciphers, we propose to encrypt every data block with a different secret key. In an attempt to

[2] MAC (message authentication code): a code for authentication and data integrity

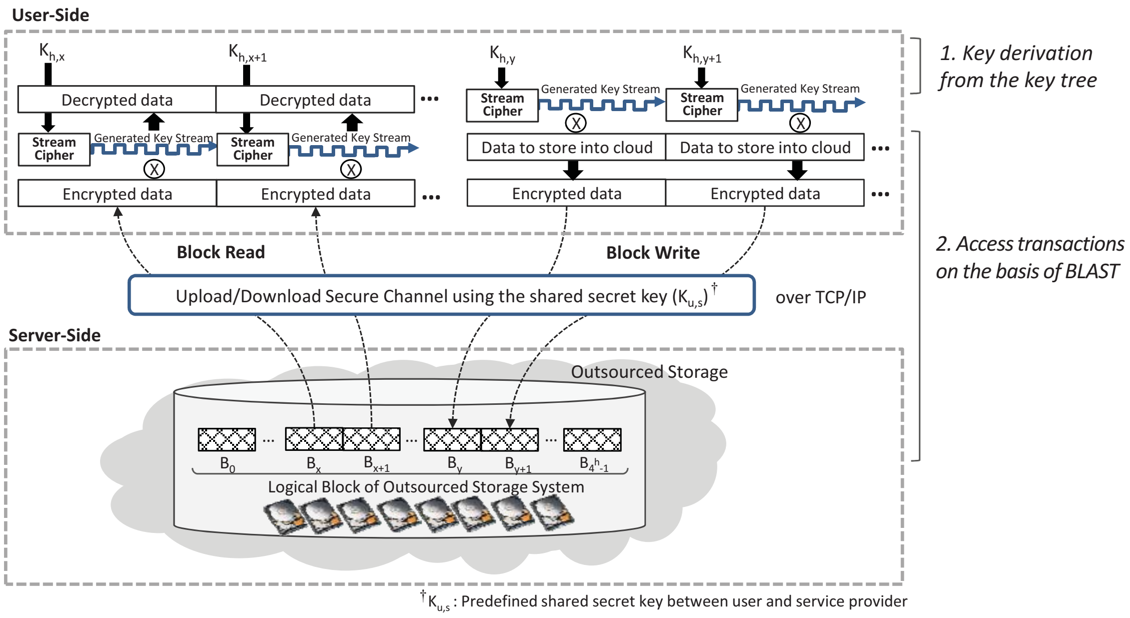

User-Side
$K_{h,x}$
$K_{h,x+1}$
Decrypted data
Stream Cipher
Generated Key Stream
Encrypted data
$K_{h,y}$
$K_{h,y+1}$
Data to store into cloud
1. Key derivation from the key tree
Block Read
Block Write
2. Access transactions on the basis of BLAST
Upload/Download Secure Channel using the shared secret key ($K_{u,s}$)[†]
over TCP/IP
Server-Side
Outsourced Storage
$B_0$ $B_x$ $B_{x+1}$ $B_y$ $B_{y+1}$ $B_{4^h-1}$
Logical Block of Outsourced Storage System
[†]$K_{u,s}$ : Predefined shared secret key between user and service provider

**Figure 3. Overall Transaction of BLAST based Storage**

provide an efficient key generation mechanism, we propose to adopt a key derivation method based on a one-way hash function, in this study SHA-1.

The basic principle is to generate keys for data block encryption through a hierarchical tree. Every key in the hierarchy can be derived by swapping the subsection of its parent node and putting it into the one-way hash function, as shown in Figure 2. This approach uses a one-way hash function (SHA-1) as a type of quad tree, implying that one parent owns a maximum of four child nodes. If a user outsources their storage, which consists of *n* blocks, the user can build a quad-tree with level $h$ $(4^{h-1} < n < 4^h)$. Through the tree, the user can derive all of the keys, $K_{a,b}$, derived from the root key $K_{0,0}$. The first index, *a,* of the key represents its level in the tree, and the second index, *b,* represents its sequence in the level. For any node $K_{a,b}$ in the hierarchy, its four child keys ($K_{a+1,4b+0}$, $K_{a+1, 4b+1}$, $K_{a+1, 4b+2}$, $K_{a+1, 4b+3}$) can be calculated by swapping the subsection of the parent key and then applying the hash function. Through repeatedly applying this procedure, a node can calculate the keys of all of its descendants. For example, for a hierarchical tree of level *h*, the sequence numbers are from *0* to $4^h$*-1*. When level *h* of the hierarchy is reached, the hash results (leaf nodes, $K_{h,x}$) will be used as keys to encrypt the corresponding data blocks ($B_x$). In this way, the user will need to maintain only the root key ($K_{0,0}$) of the hierarchical tree. The cost of this approach is the calculation of the one-way functions during the key derivation process. The experiments of this study show that even thin clients (*Intel A110 800MHz*) can accomplish these calculations very efficiently in less than *0.2ms* for *256GB* of outsourced storage (Block size: *16KB*). The system can adjust the parameters of the hierarchy to balance the storage and computation overhead during the information access storage. For example, if the user sets the child number from one parent node to *8*, *16* or *32*, it is possible to derive keys without deep hierarchies; each will then be used to encrypt a part of the outsourced data.

## C. Data Access Procedure based on BLAST

The overall transaction on the basis of the proposed BLAST based storage system is presented in Figure 3. After the generation of the key tree, the user can upload/download their data after encrypting/decrypting it, which is read or written in a block unit, via the bitwise streaming ciphers with the key stream sequence derived from the hierarchical tree. In the case of normal streaming ciphers-based secure storage, whenever a user wants to access a specific data block from outsourced storage, the generation of key streams must be done to decrypt the data, which incurs considerable overhead on a thin client, as the user has to run the underlying key stream generator forward from the initial state of the secret key up to the internal state of the corresponding block. In the proposed BLAST system, however, the generation of the corresponding key is not a burden on thin clients, as the key can be derived by applying a certain number of hash operations from the key tree. The user only needs to keeps the root key of the user's outsourced storage. In addition, the user can cache the derived keys for repeated access to the specific data block, implying that there is no need to derive the key sequence whenever accessing the data block in the future. In order to provide confidentiality, it is assumed that the service provider will conduct over-encryption when it sends data blocks to the user. To conduct this operation, the service provider and users are assumed to communicate with each other using one additional stream cipher with the predefined shared secret key, $K_{u,s}$, for the over-encryption. With these

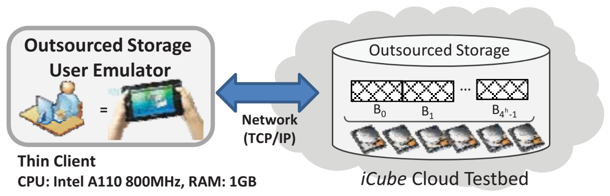


**Figure 4. Experiment Environment**

assumptions, the data access procedures such as block read, block write, block deletion, and block appending work as follows:

- **Block Read:** The user will send a request index to the service provider. When the service provider receives this packet, it can verify the users' authority using the predefined MAC. Subsequently, the service provider will retrieve the encrypted data blocks and send it through the secure channel.
- **Block Write:** When the user needs to write $B_y$, the user needs to modify the y^th block of the outsourced data. The user then will use $k_{h,y}$ to encrypt the data block and send it over the secure channel, and the service provider will store the block in the y^th block of the outsourced data.
- **Block Deletion:** When the user needs to delete a block from the outsourced storage, the user will use a special control block to replace the block. The special block, filled with zeros, will be encrypted by the corresponding key and transmitted over the secure channel. The service provider then replaces the block on the outsourced storage system.
- **Block Appending:** The user may need to store additional data and put it on the outsourced storage. In order to provide scalability of the storage capacity, the right-most child node, at the level $h$ in the tree, is reserved as an extension node. Later, the key can be used to derive new extended keys for new data blocks.

## III. Experiments and Evaluation

This section presents the performance results obtained with the proposed prototype implementation of the BLAST-based secure storage system. First, the overall experimental environment is demonstrated, and the scheme is then compared with Block- and Streaming cipher-based secure storage systems which are enhanced with a block cipher and a stream cipher, respectively, to verify the capability of the proposed scheme.

### A. Experiment Environment

To evaluate the performance characteristics of the BLAST based storage system, a cloud user emulator (CPU: *Intel A110 800MHz*, RAM: *1GB*) connected with 100Mb Ethernet and a cloud storage system were constructed on the basis of BLAST within a real cloud computing service known as *iCube Cloud*[3]. Figure 4 shows the overall experimental environment. The emulator is to compare the performance of our scheme with AES-CTR mode as a block cipher and SEAL3.0 as a stream cipher for the secure storage systems. The lengths of key are 128 bit and 160 bit respectively. However, we note that our scheme can be realized using any kind of streaming ciphers such as SEAL, Rabbit, and RC4 etc. The emulator involves reading and writing multiple 4KB – 16MB files with sequential and random accesses. Since main objective of the experiment is to evaluate our scheme with block and stream cipher-based secure storage systems, we set the subordinate factors such as disk access and network delay aside from our performance evaluation.

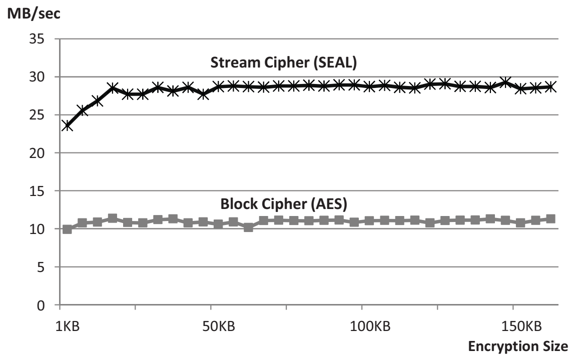


**Figure 5. Encryption Throughput of Block/Stream Ciphers**

### B. Stream Cipher vs. Block Cipher

Figure 5 shows the encryption throughput of the AES as a block cipher and SEAL as a stream cipher. The encryption is performed with different data block sizes. With a small block size, SEAL tends to perform slightly better than AES. However, as the data block size increases, the throughput of SEAL becomes far better than that of AES. This shows that SEAL is more efficient than AES in encrypting large data blocks. In spite of the efficiency of SEAL, it is commonly considered that SEAL is not suitable for a storage system, because in order to decrypt a certain block, one generally has to run the underlying key stream generator forward from the initial state of the secret key up to the internal state of the corresponding block.

### C. Block Accessibility Evaluation

The performance of the secure storage system in terms of its throughput and its consumption of processing resources is an important factor before it can be deployed as an actual storage system. First, the throughput of the storage system based on BLAST was analyzed in a comparison with that based on block cipher (AES) and stream cipher (SEAL). We measured the throughput while varying the start address of the operation and the length of the access data for the storage system operations. While one parameter was varied, others were fixed at set values in all experiments. The

[3] http://www.icubecloud.com: Cloud service platform & testbed of this work

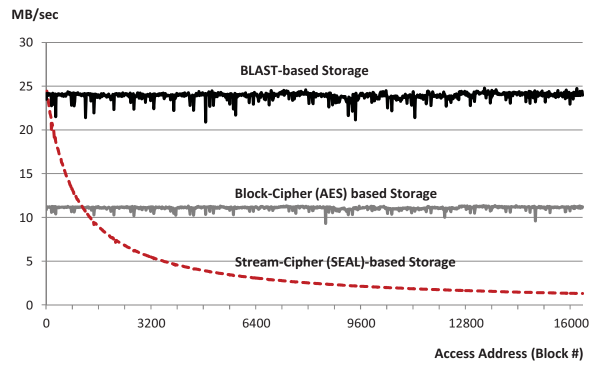
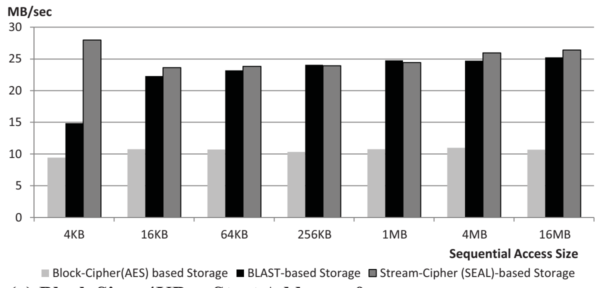


**Figure 6. Storage Transaction Throughput with Varying Start Address**

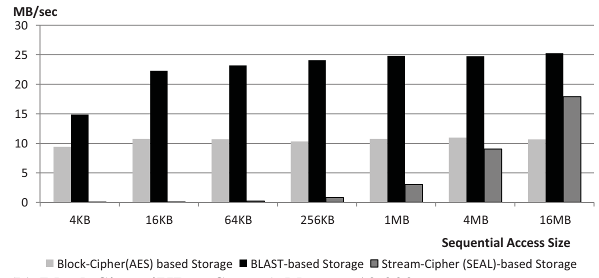
experiments are configured that the access ratio of read to write is 50% to 50%.

- **Access while Varying the Start Address:** To show that the proposed scheme overcomes the weakness of random access in the stream cipher, an experiment was designed with storage access operations that took place while the start address was varied from *0* to *16,000*. In this experiment, it was assumed that the block size of BLAST was *4KB* and that the length of the access data was *1MB*. Consistent throughput regardless of the start address indicates the block accessibility with *O(1)*-time planarity which is suitable for random access. Conversely, a decreasing throughput curve implies that a penalty on performance increases with *O(n)*-time planarity which makes random access unavailable. Figure 6 verifies this information. AES, a block cipher with the capability of random access, shows consistent throughput while the start address of the operation varies. In contrast, SEAL, a stream cipher without the capability of random access, shows a decreasing curve. Based on this verification, the proposed algorithm, BLAST, is shown to have the capability of random access, as the algorithm also shows the consistent throughput. In addition, BLAST outperforms AES, which represents a block cipher. This is clear given that AES requires a more complicated process to encrypt and decrypt, as shown in Figure 5.

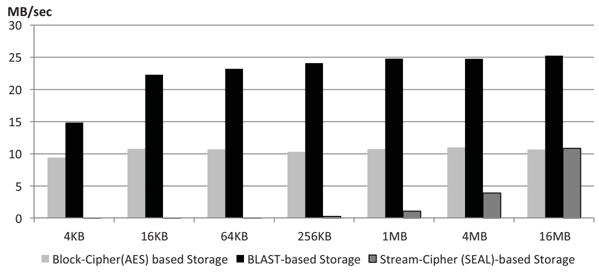
- **Access while Varying the Length of the Data:** According to the length of the data used for accesses in the experiments, the throughput can be changed. For the experiment in Figure 7, the block size was set to 4kb and the start address was varied from *0* to *30,000*. Subsequently, the throughput was measured while varying the length from *4KB* to *16MB,* as since most file system accesses are within this range in actual environments [16]. In Figure7-(a), the graph shows the throughput when the start addresses of accesses were set to *0*. SEAL-based storage performs slightly better than BLAST since there is no overhead of key derivation. However, this phenomenon was reduced and BLAST outperforms SEAL in Figure7-(b) and (c) which start addresses were set to 10,000 and 30,000, respectively,



**(a) Block Size: 4KB,  Start Address : 0**



**(b) Block Size: 4KB,  Start Address : 10,000**



**(b) Block Size: 4KB,  Start Address : 30,000**

**Figure 7. Throughput with Varying Sequential Access Size**

since SEAL suffers from the penalty of O(n)-time planarity which mentioned in previous section. In the perspective of the length of the data, the stream cipher, SEAL experiences better throughput when the length of the data becomes longer. However, it was clear that BLAST outperforms in terms of the throughput of normal block/stream cipher-based storage, AES and SEAL respectively. BLAST does not change much following an increase in the length of the data within the aforementioned range. The throughput of SEAL may exceed that of BLAST over the range in this experimental configuration, however, the cross point will vary proportional to the start address because of the reason shown in Figure 6. Thus, the proposed scheme, BLAST, is clearly more than suitable compared to normal block (AES) and stream (SEAL) cipher based secure storage system.

## *D. Key Derivation Overhead*

In the proposed secure storage system, users need to derive multiple keys to access the corresponding data blocks. To measure the key derivation overhead, the time ratio of the key derivation time to the data access delay, which includes the storage access and data encryption/decryption process, is shown in Figure 8. In the short data access case,

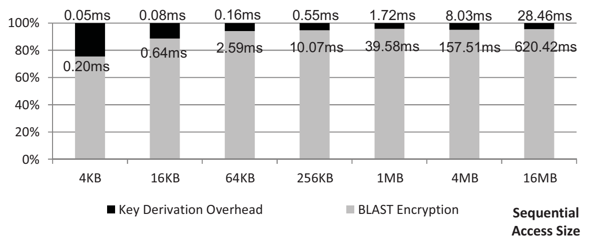


**Figure 8. Key Derivation Overhead with Varying Access Size**

the key derivation delay is at most *0.05ms*. In addition, a larger data access progressively reduces the portion of time consumption for the derivation of the key. Hence the proposed approach introduces very limited key derivation overhead with block accessibility.

Secondly, the maximum key derivation delay for a specific block while varying the total storage size is shown in Figure 9. The block size was set to *4KB*. Subsequently, the maximum key derivation delay for a specific block (no cache effect) was measured while varying the total storage size from *4GB* to *16PB*. Even in a Peta-scale storage system on the basis of the proposed BLAST, it requires acceptable operation overhead less than *0.3ms*. In addition to that, the delay can be mitigated by applying a cache of the derived keys.

## *E. Safeness of BLAST*

In this section, the security strength was analyzed to confirm the safety of the proposed scheme. It was assumed that the underlying cryptography of stream cipher and SHA-1 were invulnerable with regard to message secrecy and integrity. In terms of the security of the key derivation scheme, this study refers to other work [13, 14], as the proposed scheme uses the identical principles and safety concepts. On the other hand, it was assumed that the service provider will attempt to gain access to the plaintext of the stored data. In accordance with this assumption, the service provider can analyze the cipher text by searching for statistical similarities between different pieces of the cipher text or some sequences that occur more than others. If the cipher text is purely random without statistical irregularities, the provider would have to resort to an exhaustive brute-force key search attack. The key stream frame of the key tree on the basis of BLAST is generated by swapping the subsection of its parent node and putting it into the one-way hash function, SHA-1. As it is known that the secure hash generates an approximately random key stream, the key stream unit of the proposed scheme is approximately random. If a cryptanalyst does not have the root key of the key tree, a brute-force key search must be used. In a brute-force attack, the key search space is as large as the key space of the stream cipher. Therefore, the proposed key generation scheme has a larger key search space than the single stream cipher.

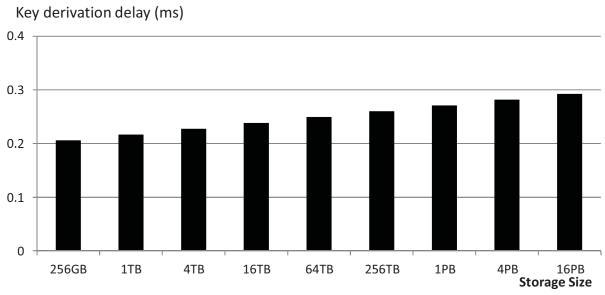


**Figure 9. Key Derivation Delay with Varying Storage Size**

## IV. RELATED WORKS

Recently, outsourced storage system enhanced with security has been studied and developed in the research area of cloud computing. In this section, we briefly discuss the pre-existing secure storage systems.

With a respect of secrecy in the filesystem layer, many researchers tried to improve the cryptographic filesystem. As a pioneer research, CFS [20] is suggested to provide secure storage mechanism for a general filesystem. In an attempt to provide networked secure storage system, TCFS [21] is proposed which is an extended from NFS. Since the filesystem is on the side of the trustable user, it can securely transfer the encrypted data to the outsourced storage through network. CryptFS [22] concentrates on implementation to overcome portability. Being kernel-resident, it can be on top of various filesystems. In order to provide better integrity checking, RAND-EINT and COMP-EINT [23] are proposed. For the integrity checking, the user should maintain some information such as hashed values to check the integrity of the outsourced storage. These proposed techniques optimize the maintenance overhead of the information.

In an attempt to deploying encryption schemes in secure storages, there have been several works to improve the cryptographic filesystem. They rely on existing encryption algorithms. In the beginning of the cryptographic file system, DES [4] (data encryption standard) which is a symmetric encryption algorithm and a block cipher with the 56 bits key had widely used. Since DES has reached expiration, AES[5] which is the advanced version of DES was newly adopted in the cryptographic filesystem for the higher security. The main reason of employing the block ciphers into secure storage systems is due to its block accessibility feature. Block cipher-based secure storage systems such as those in a recent study [5-7] can provide block accessibility but have relatively high operation complexity. The stream cipher-based secure storage system, on the other hand, can have lower cryptographic operation overhead only for sequential access patterns due to the missing block accessibility function.

[4] **DES(Data Encryption Standard)** is a block cipher on the basis of symmetric key operation.

[5] **AES(Advanced Encryption Standard)** is an encryption standard on the basis of symmetric key operation (FIPS 197).

Our approach is to provide computation efficiency by enabling to apply the stream ciphers into storage system without compromising of the confidentiality. Also, it can be accessible in random manner. To the best of our knowledge, none of the secure storage has tackled to deployment stream ciphers into secure storage system. We present stream-cipher based storage system which is to provide a computationally efficient secure outsourced storage system.

## V. CONCLUSION AND FUTURE WORK

The task here was to provide a full-fledged secure and computation efficient storage solution tailored for a cloud computing environment. To accomplish this task, a block accessible encryption mechanism based on streaming cipher, termed BLAST was presented to enable construction of a secure storage system based on a stream cipher rather than on a block cipher. The experimental results show that the proposed scheme is more time efficient than a normal block/stream cipher-based secure storage system. Moreover, the proposed scheme can be adapted to relevant work we described without changing the basis of their system. Still there is room for potential improvement in the proposed design and implementation that we did not care in this paper. The proposed key derivation scheme should include more sophisticated caching mechanisms in order to reduce the key derivation overhead since multiple accesses of the user to the outsourced storage triggers repeated hash operations with same parameter. As the next step of this study, we are exploring more appropriate integrity and access control mechanism for multiple users in BLAST based secure storage system.

## ACKNOWLEDGMENT

The authors wish to thank their anonymous referees for all of their invaluable comments and suggestions. The authors would also like to thank *NexR Co. Ltd* for their help in conducting some of the experiments in this study. The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. A1100-0901-1710.

## REFERENCES

[1] Amazon.com, Inc., “Amazon Simple Storage Service”, http://aws.amazon.com/ec2/

[2] Azure Service, “Microsoft, Windows Azure Platform”, http://www.microsoft.com/windowsazure/

[3] Ristenpart T., Tromer E., Shacham H, and Savage S., “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds”, 16th ACM Conference on Computer and Communication Security (CCS) 2009.

[4] P. Prasithsangaree and P. Krishnamurthy, “Analysis of Energy Consumption of RC4 and AES Algorithms in Wireless LAN”, IEEE GLOBECOM 2003.

[5] A. Singh and L. Liu, "Sharoes: A data sharing platform for outsourced enterprise storage environments", IEEE Data Engineering, pp.993-1002, 2008.

[6] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang,and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage", USENIX FAST, pp.29-42, 2003.

[7] R. Pletka and C. Cachin, "Cryptographic security for a high-performance distributed file system", IEEE Confer-ence on Mass Storage Systems and Technologies, pp. 227-232, 2007.

[8] A.Yun, C.Shi, and Y.Kim, “On Protecting Integrity and Confidentiality of Cryptographic File System for Out-sourced Storage”, ACM Workshop on Cloud Computing Security, 2009.

[9] J.Li, M. N. Krohn, D.Mazieres, and D.Shasha, “Secure untrusted data repository (SENDR)”, USENIX OSDI pp.121-136, 2004.

[10] W.Wang, Z.Li, R.Owens, and B.Bhargava, “Secure and Efficient Access to Outsourced Data”, ACM Workshop on Cloud Computing Security 2009.

[11] C. Wang, Q. Wang, K. Ren, and W.Lou, “Ensuring Data Storage Security in Cloud Computing”, IEEE IWQoS, 2009.

[12] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: management of access control evolution on outsourced data", Conference on Very large databases (VLDB), pp.123-134, 2007.

[13] M. Atallah, K. Frikken, and M. Blanton, “Dynamic and Efficient Key Management for Access Hierarchies”, ACM Conference on Computer and Communication Security (CCS) 2005

[14] H.Chien and J. Jan, “New Hierarchical assignment without Public Key Cryptography”, Elsevier Computers and Security, Vol. 22, No. 6, pp.523-526, 2003

[15] S. Gaonkar, K. Keeton, A. Merchant, W. Sanders, "Designing Dependable Storage Solutions for Shared Appli-cation Environments," IEEE Transactions on Dependable and Secure Computing, 16 Jul. 2008

[16] Tanenbaum, A. S., Herder, J. N., and Bos, H. 2006. File size distribution on UNIX systems: then and now. SI-GOPS Oper. Syst. Rev. 40, 1 (Jan. 2006), 100-104.

[17] Reeds J. A. ; Wwinberger P. J., “File security and the UNIX system crypt command”, AT&T Bell Laboratories technical journal , 1984, vol. 63, no8, part 2, pp. 1673-1683

[18] RSA Laboratory, “RC4 Specification”, Technical Report 1994, RSA Press

[19] P. Rogaway and D. Coppersmith, "A Software-Optimized Encryption Algorithm", Fast Software Encryption, Cambridge Security Workshop Proceedings,Springer-Verlag, 1994, pp. 56-63.

[20] Blaze, M. 1993. A cryptographic file system for UNIX. In Proceedings of the 1st ACM Conference on Computer and Communications Security (Fairfax, Virginia, United States, November 03 - 05, 1993). CCS '93. ACM, New York, NY, 9-16.

[21] Tcfs E. Mauriello, “TCFS: Transparent Cryptographic File System”, Linux Journal, 40, Aug. 1997.

[22] E. Zadok, I. Badulescu, and A. Shender, “Cryptfs: A stackable vnode level encryption file system”, Technical Report CUCS-021-98, University of Cal-ifornia at Los Angeles, 1998

[23] Opera integrity A. Oprea, M. K. Reiter, and K. Yang, “Space-efficient block storage integrity” In Proc. Network and Distributed System Security Symposium (NDSS). ISOC, 2005.

[24] M. Boesgaard, M. Vesterager, and E. Zenner., "The Stream Cipher Rabbit. New Stream Cipher Designs", Lecture Notes in Computer Science, 4986:69-83, 2008

[25] NexR. Co. Ltd., “iCube Cloud Computing and Elastic-Storage Service”, http://www.icubecloud.com, 2010

[26] K-W Park, S. K. Park, and K.H Park, "THEMIS: Towards Mutually Verifiable Billing Transaction for Cloud Computing Environment," IEEE International Conference on Cloud Computing, 2010