KaaSP: Keying as a Service Provider for Small and Medium Enterprises Using Untrusted Cloud Services

William Aiken
Department of Information Sciences and Technology (IST), Pennsylvania State University
3000 Ivyside Park
Altoona, PA, 16601
+1-814-949-5000
wva5029@psu.edu

Jungwoo Ryoo
Department of Information Sciences and Technology (IST) Pennsylvania State University
3000 Ivyside Park
Altoona, PA, 16601
+1-814-949-5000
jryoo@psu.edu

Hyoungshick Kim
Department of Software
Sungkyunkwan University
2066 Seobu-Ro, Jangan-Gu
Suwon, Gyeonggi-Do, Republic of Korea
+82-31-299-4324
hyoung@skku.edu

ABSTRACT

Cloud computing provides a framework for allowing remote and nearly instantaneous access to data and resources from any location in the world with an Internet connection. However, it faces privacy concerns since cloud service providers can also access user data on their storage. Although several encryption services and applications were introduced for personal users, it is still questionable whether such services can effectively be deployed for enterprises due to their lack of scalability. We propose a new access control system that incorporates encryption, based on access via a third-party key management service.

The proposed system introduces a new entity named a Keying as a Service Provider (KaaSP) to more securely provide a data encryption service. In our approach, data encryption keys are generated through a negotiation with the KaaSP which would not have access to all key parts. Therefore, even if petitioned by a powerful adversary such as a law enforcement organization or breached by an attack, the data could not be leaked. Moreover, user data on the cloud storage can be protected from access attempts made by a lost device controlled by an unauthorized user since a lost device's credential for authentication can instantly be revoked. Additionally, the controlling organization can seamlessly edit access credentials via this cryptographic framework.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Cryptographic controls; H.2.7 [**Database Administration**]: Security, integrity, and protection

General Terms

Security, Design, Management

Keywords

Cloud computing; Key management; Domain management; AONT; All-or-Nothing transform

IMCOM '15, January 08 - 10 2015, BALI, Indonesia

ACM 978-1-4503-3377-1/15/01…$15.00
http://dx.doi.org/10.1145/2701126.2701206

1. INTRODUCTION

Cloud computing holds promising potential for the future of the information technology. It allows for simple, inexpensive, manageable, scalable, and portable computing resources with less overhead costs. Many startup businesses have been born in the cloud, and several large businesses have fully adopted a cloud environment. However, despite its advantages, there is still reluctance to integrate completely into the cloud, and security and privacy issues are often mentioned as the primary concern. As Moore's Law continues, we are likely to see mobile devices able to process larger key sizes locally in real-time. Larger keys, however, will not solve all the problems because as more devices can handle sufficiently large keys, and more employees migrate their work to mobile devices such as smart phones, the total number of keys that an organization has to manage may continue to increase. In other words, even though more devices will be able to use longer key lengths, the total number of keys may also become unmanageable for businesses as each employee's device is given its own unique key. Our solution aims to resolve many of the problems associated with key creation, key management, and key renewal as well as provide real-time, measurable audits in order to comply with international standards for information systems such as ISO 27001 [5].

Our system is based on the communication between three entities to decrypt the cloud-stored data. These three entities are the organization, the end user, and a third-party key manager, which we refer to as the Keying as a Service Provider (KaaSP). Our research is largely based on the prior work of a system named EnCloud for providing end-to-end encryption between cloud applications [6]. The EnCloud system was designed around the concept of registering a Domain Client (DC) to a Domain Manager (DM). The DC is responsible for all encryption and decryption locally, while the DM is responsible for managing the registration and revocation of its clients, therefore, the sharing of keys with the clients. Their research demonstrates that the encryption/decryption delay for privacy in a private cloud is negligible compared with communication overheads, and an end-user of cloud services using EnCloud could securely encrypt data against powerful adversaries like governmentally-funded organizations.

One may ask why a new approach like ours is necessary when cloud service providers (CSPs) could offer a similar type of service. Our response is that CSPs are not in the business (or sometimes legally able) to provide privacy against powerful (governmental) adversaries, and some security vulnerabilities that could compromise data across all CSP platforms, hypervisors, etc.

may never be discovered. A zero-day attack on a CSP authentication system with just one compromised key could possibly result in catastrophic data loss for an enterprise. There is also a growing concern that many CSPs are already allowing powerful adversaries to access user information via back-doors [4]. In our proposed system, the third party KaaSP would simply manage the retrieval of and access to a part of the decryption key before decryption takes place locally.

Our system builds upon EnCloud [6] to focus on the challenges that small- and medium-sized enterprises (SMEs) face. EnCloud requires the manual removal of a lost (or stolen) device from the DM to prevent the DC running on the device from having access to personal data on the cloud. This may, however, present a serious problem to organizations. Since decryption time is negligible, a stolen DC could be used to download and decrypt cloud data long before the lost device is manually revoked.

Therefore, we propose a framework that improves the security strength of key management rather than increasing encryption strength. This way, a business can instantly revoke a lost device by changing access credentials in the DM in order to remove access to its key and its privileges to access cloud data. Our system will transparently require authentication at every instance of the use of the key through an all-or-nothing (AONT) framework to achieve an auditable cryptography transform method for users and devices [1]. Unless all 3 parties (the DM, the DC, and the KaaSP) are communicating and authenticating, the cloud data cannot be decrypted. In this way, the organization will have control over access to any highly-sensitive data, and the organization will be able to instantaneously revoke privileges internally. Additionally, the organization can meet demands from international auditing standards that require access and encryption policies, which we discuss in our implications section.

Some organizations may prefer a 2-party secret-sharing framework in which an end user could decrypt data either when connected to the internal network (via the DM) or on the external network (via the KaaSP) instead of facing additional network overhead by requiring communication among three parties in our scheme. A 2-party secret-sharing framework is possible within our framework because essentially an AONT is a secret sharing framework where the number of players required to decrypt the message is equal to the number of total players. To account for less than all parties being present, we must account for all other possible combinations of parties. In this case, that results in 2 possible combinations:

1) DC + KaaSP
2) DC + DM

Assuming a symmetric key length of 256 bits, the total increase in file size over the AONT requiring all three parties per file stored in this manner is 256 additional bits plus any padding added during the Elliptic Curve Cryptography (ECC) encryption. This occurs because each combination of parties (i.e. DC + KaaSP and DC + DM) has its own asymmetric key pair. Therefore, the symmetric key will be encrypted and attached to the file twice in this case. We elaborate on this in the AONT Framework Description section.

Our system will, therefore, attempt to meet the demands of the current threat environment of powerful adversaries while providing a robust key management system based around small- and medium-enterprises. We aim to protect SMEs' private data stored in the cloud so that the adversary only knows the presence of the encrypted data and their physical characteristics (e.g., creation time, size, etc.) but not the contents. Through the KaaSP system, SMEs will be able to control, audit, and account for every end user's keys. This way, an organization can outsource all of its computing resources while still remaining in control of its security measures.

In this paper, we will demonstrate our system as a proof-of-concept for the possibility of a third-party Keying as a Service Provider. We discuss the existing frameworks on which our model builds or in which it would effectively operate. Most importantly, we describe a step-by-step explanation for how the system will operate in practice. Finally, we define the limitations, implications, and future research that will accompany our proposed system.

2. RELATED WORK

Encryption in the cloud is a fast-growing field. As mentioned, portable devices are becoming more popular, and computing power is increasing. Much research has been done to quickly and securely access encrypted, cloud-stored data. For example, Moghaddam et al. proposed a data encryption service where keys and data are stored separately into a Key Cloud Server and a Data Cloud Server respectively [10]. Their work compares various cryptographic schemes based on RSA (Rivest, Shamir and Adleman) as well as varying key sizes from 512 to 3072 according to time and security constraints. Our system, will, however, focus mainly on the implementation of a key cloud server, using ECC at a key size of desired length by the DM.

Another similar architecture was proposed by Zhao et al., which is based on developing homomorphic encryption [9]. In their model, the user connects to a management system which is related to three additional parties: a key management module, a data processing system, and a data storage system. They proposed that a trusted third party could process the ciphertext data directly for efficient and secure queries of large amounts of ciphertexts. Our system could easily be worked into such a scheme, where the KaaSP acts as the key management and authentication module of Zhao et al.'s security architecture.

Ghebghoub et al. offered a security model based on Organization Role-Based Access Control (ORBAC) [8]. Their solution is designed to allow the access control to be regulated by the owner's data by assigning each resource to its own secret key. Their research goal was to offer a new layer between different users and the owner's cloud data. However, it assumed that there does exist a dedicated "owner" of the data, but in some cases of shared resources, no one person may be considered an owner but merely the creator who is of equal status with the others who need access to the resource. Nonetheless, their research shows that ORBAC in accessing cloud data in a context lessens risks, provides trust, increases performance, and optimizes resource use.

Hei and Lin proposed a scheme where certain parts of a file are encrypted with different keys [7]. They provided access-control on a file-by-file basis. The entire file can be decrypted by the users, but the users only receive the parts they are authorized to access. Hei and Lin proposed a unique key based on a healthcare provider's license, role, and attributes, and the owner or patient can grant access on a by-field basis. They stated that ciphertext-policy-attribute-based encryption (CP-ABE) allows for this fine-grained access control while RSA encryption could not, but CP-ABE yielded results of 100 to 1000 times slower than RSA.

Much research has been done relating to many more schemes for key creation, cryptography scheme comparisons, attribute distribution, etc. However, we believe that the field is lacking in a concrete proposition appropriate for small- and medium-sized organizations. We extend the EnCloud system [6] which was designed for end-to-end encryption between cloud applications by introducing a service component named KaaSP, to reduce the burden of key management for SMEs that either fail to understand or cannot afford the complexity behind their own cryptography and key management systems. There already exists open source software that allows for multiple keys per access to one file. That is, both user A's key and user B's key can decrypt their corresponding symmetric keys with the same result. For example, GNU Privacy Guard (GNUPG) supports this [3]. Our solution allows an organization to easily add and remove devices and privileges while outsourcing the actual management of both data and keys to trusted third parties.

3. OVERVIEW OF PROPOSED SOLUTION

We propose an end-to-end encryption system for SMEs using an All-Or-Nothing Transform (AONT) scheme which utilizes a third-party key manager. The concept of AONT was first proposed by Rivest in 1997 [1]. In short, x_1 and x_2 are used to encrypt plaintext P into AONT(P). If, for example, x_2 is compromised but not x_1, the attacker is no closer to discovering P than if he had neither x_1 nor x_2. Our proposed system uses AONT as a key functional component. There are three parties in our system: 1) the Domain Client (DC), also known as the user, 2) the Domain Manager (DM), which is on a secure network to which the DC belongs, and 3) the Keying as a Service Provider (KaaSP), which operates as an independent third party.

At the most basic level, access to the key will be controlled by all three parties. The first party would be the DM, preferably an always-on desktop computer secured with the enterprise; the second party would be the DC, a mobile device that would need access to cloud-based data from any location; and the third party would be a third-party Keying as a Service Provider (KaaSP). However, our proposed system does allow for a threshold of at least 2 parties with minimal overhead. In this way, the controlling organization is more able to tailor the access control system to its specific needs.

For SMEs, establishing, running, and maintaining a user-authentication scheme before downloading the cloud data defeats the purpose of outsourcing computer operations to a cloud provider. As a result, we believe an impartial third-party should hold part of the access to the decryption key from a remote location that requires authentication before retrieval. Unlike the DM and the DC, the KaaSP needs not be trusted by an organization. Even a complete compromise of the entire KaaSP database would reveal only part of the data encryption key to the attackers. This is because none of the parties without the others have any meaningful information about the decryption key which is needed to access the encrypted data stored on the cloud storage.

4. AONT FRAMEWORK DESCRIPTION

In this section, we will provide the details for our proposed encryption framework. Each of the three parties has a randomly generated string. The DM has s_1.; The DC has s_2.; The KaaSP has s_3. DC assembles all three strings and XORs them together, finally hashing the result to get SK, which will function as the private key in an Elliptic Curve Cryptography algorithm. SK is then used to securely decrypt the data encryption key DK encrypted by the corresponding public key PK. The proposed SK generation method based on XORs is the simplest form of secret sharing schemes. We can replace this operation with an advanced secret sharing scheme according to the system requirements. For example, a (t, n)-threshold [12] can be used to achieve fault tolerance in some environments; with this scheme, even in the unlikely event where one entity refuses to share his portion of the secret, the symmetric decryption key may still be recovered.

We note that the use of public key has an important advantage over the symmetric key alone. When we use AONT for public key encryption, the key-assembly operation is only required for decrypting the cloud data. The data upload can be locally processed without the key-assembly operation, using the stored public key PK. Therefore we propose using a public key cryptosystem such as ECC rather than a symmetric key cryptosystem alone because the assembly of a key may be very expensive in some situations.

In the proposed AONT system, the user (DC) would have to authenticate itself to both the company's DM and the KaaSP in order to have access to the decryption key. The DM and the KaaSP would send their portions of the pseudo key to the DC where a reconstruction of SK would then be possible. Then the DC would be able to use SK to decrypt DK, which could then be used to decrypt cloud-stored data. No party has enough information of the SK for even a partial reconstruction, hence the name all-or-nothing, and DK resides only in its encrypted form until decrypted by SK. Unlike a traditional cloud-client environment where the CSP handles encryption, in our approach, the user could specify the exact encryption method to avoid back-door encryption holes that a CSP may have. The DK will be used to encrypt the cloud data before being encrypted by the corresponding public key PK. The encrypted DK will then be appended to the data. These encrypted DKs would be in number equal to the number of unique access credentials used to retrieve the source data. With a strong ECC key of 256 bits and 100,000 unique keys, that would only require additional 32 kilobytes, a near negligible amount of additional data when encrypting documents in the size of even megabytes.

We assume that the DM and the DC are trusted and not under the control of the adversary. We also assume that there are pre-established secure channels between the DM, the DC, and the KaaSP. These channels can be established by various standard cryptographic protocols. For example, the secure channel between the DM and the DC can be constructed by using a common secret such as user password [6].

For encryption schemes, we propose using ECC as the encryption scheme to encrypt the symmetric key(s) attached to the encrypted data. We note that ECC is more appropriate than RSA for our purpose. As we mentioned above, the corresponding public key PK should be generated from a random private key SK generated by a hash algorithm. Unlike ECC, in RSA, it is not computationally feasible to generate a public key in such a manner. Furthermore, one advantage of the ECC over the RSA is that ECC offers almost the same cryptographic strength as RSA but in a shorter key length. For instance, the 160-bit ECC and the 1,024-bit RSA are comparable in cryptographic strength [2]. Therefore, ECC has the advantage of using lesser bandwidth and requiring less storage. Also, ECC is believed to be more secure. The highest-bit ECC scheme broken was at 112 bits by the Laboratory for Cryptographic Algorithms [11]. Because of these factors, we believe that ECC is the best candidate for the proposed system. For symmetric encryption, we propose using AES which is a (de facto) global standard for data encryption.

We will describe a step-by-step approach to explain how the proposed system works for basic operations such as data encryption and decryption. We will use the following symbols and notations:

$E_K(X)$: Encrypt X with a key K; if E is a symmetric encryption algorithm, $E_K(X)$ is decrypted by the same key K; if E is a public key encryption algorithm, $E_K(X)$ is decrypted by the public key K's corresponding private key,

CSP: Cloud Service Provider,

KaaSP: Keying as a Service Provider

DC: Domain Client (e.g., mobile device)

DM: Domain Manager (e.g., desktop computer)

DK: Data encryption key

SK: Private key used in ECC

PK: Public key used in ECC

□: XOR

S: Secret generated by AONT (here, using an XOR operation)

s_x: Randomly generated key part, used in AONT.

4.1 Key Configuration

For the initial key configuration, the following steps are processed at the DM, the DC and the KaaSP, respectively:

Step 1) DM, DC, and KaaSP generate their own secrets s_1, s_2, and s_3, independently.

Step 2) After successful logins, DC requests s_1 and s_3 to DM and KaaSP, respectively.

Step 3) Via secure channels, DM sends s_1 to DC; KaaSP sends s_3 to DC.

Step 4) DC generates $S = s_1 \square s_2 \square s_3$ and then hashes S using a securing hash algorithm to yield SK which will act as DC's private key by means of Elliptic Curve Cryptography.

Step 5) DC generates the corresponding PK from the given SK.

Step 6) DC discards s_1, s_3, and SK by using a secure deletion method and stores s_2 alone. Therefore, DC holds s_2 and PK.

4.2 Data Upload

For uploading the data D to the CSP, the following steps are processed at the DC:

Step 1) DC generates a data encryption key DK and encrypts the data D with DK to form $E_{DK}(D)$.

Step 2) DC uploads $E_{DK}(D)$ to CSP via the conventional interface provided by CSP.

Step 3) DC encrypts DK with PK to form $E_{PK}(DK)$ and then uploads $E_{PK}(DK)$ to CSP in the same manner as for the normal file upload process. Therefore, $E_{DK}(D)$ and $E_{PK}(DK)$ are stored on the cloud storage.

The data encryption key DK is used to encrypt and decrypt the cloud data. In practice, it is more preferable to use an individual data encryption key rather than the common data encryption key to encrypt all data. This is because the use of an individual data encryption key for each data can reduce the encryption/decryption overheads when the symmetric key DK is replaced with a new one.

4.3 Data Download

For accessing the data D in the CSP, the following steps are processed at the DC:

Step 1) After successful logins, DC requests s_1 and s_3 to DM and KaaSP, respectively.

Step 2) Via secure channels, DM sends s_1 to DC; KaaSP sends s_3 to DC.

Step 3) DC generates $S = s_1 \square s_2 \square s_3$ and then hashes S using a securing hash algorithm to yield SK.

Step 4) DC downloads $E_{PK}(DK)$ and $E_{DK}(D)$ at the same time.

Step 5) DC decrypts $E_{PK}(DK)$ with SK to obtain DK.

Step 6) DC decrypts $E_{DK}(D)$ with DK to obtain D.

Step 7) DC discards s_1, s_3, and SK by using a secure deletion method and stores D. Therefore, DC can successfully access D.

Some metadata or an identification scheme is needed to associate $E_{PK}(DK)$ with $E_{DK}(D)$.

4.4 Alternate 2-Party Framework

For allowing and encrypting data D with communication by only two parties, the following steps are processed at the DM, the DC and the KaaSP, respectively:

Step 1) DM, DC, and KaaSP generate their own secrets s_1, s_2, and s_3, independently.

Step 2) After successful logins, DC requests s_1 and s_3 to DM and KaaSP, respectively.

Step 3) Via secure channels, DM sends s_1 to DC; KaaSP sends s_3 to DC.

Step 4) DC generates $S_{DC+DM} = s_1 \square s_3$, $S_{DC+KaaSP} = s_2 \square s_3$. DC then hashes these two S separately using a secure hashing algorithm to yield SK_{DC+DM} and $SK_{DC+KaaSP}$ which will act as DC's private keys by means of Elliptic Curve Cryptography.

Step 5) DC generates the corresponding PKs from the given SKs.

Step 6) DC discards s_1, s_3, and SK by using a secure deletion method and stores s_2 alone. Therefore, DC holds s_2 and two public keys: PK_{DC+DM} and $PK_{DC+KaaSP}$.

DC will then follow the procedures from the 4.2 Data Upload section except DC will encrypt DK with both PKs to form $E_{PK\_DC+DM}(DK)$ and $E_{PK\_DC+KaaSP}(DK)$. DC will upload these in the same manner for the normal file upload process.

In this manner, DC will now be able to access the cloud data without being connected to DM, if the organization chooses not to allow a connectivity to its network at all times. The KaaSP handles the authentication in this method.

5. SECURITY ANALYSIS

Our proposed AONT model is designed to provide confidentiality of user data by combating both brute force attacks and man-in-the-middle attacks conducted by an adversary who can freely access data on cloud storage. The sensitive data is encrypted with a randomly and locally-generated symmetric key (DK) with a key length of at least 256 bits. The data encryption key DK is encrypted again with a public key PK so that applications with the

corresponding private key SK can only access the data encryption key DK.

Since the creation of the private key SK is locally processed at the DC and the secrets are securely exchanged via secure channels, an adversary cannot access the user data D even if the adversary can monitor communications between the DC, DM, and KaaSP.

Moreover, even if a complete compromise of DM and KaaSP were to occur, only the randomly generated key parts (s_x) would be compromised. And since, by definition, AONT results in no knowledge of the key being revealed unless all parts are known, the attackers could not reconstruct SK. This holds true even if an attack attempts a man-in-the-middle attack by intercepting communications either between the DM and DC or between the DC and the KaaSP. The DC will never send its unique key part over any network. It will always remain locally stored and accessed. Therefore, reconstructing SK by interception of messages is impossible.

A major challenge is handling the theft of a DC without knowledge of the DM or KaaSP. Even so, authentication to the DM and the KaaSP, using a shared secret by DC (like username and password) adds another layer of security, especially when utilizing the framework of secure communication between DM and DC as proposed in the EnCloud paper [6]. As a result, the end user, the DM, and the KaaSP have time to recognize either the theft of the device or the attempted unauthorized access after various failed attempts to connect.

Probably, the biggest threat to the security of this system is an insider turned deliberately malicious without any forewarning. The KaaSP system obviously allows for the revocation of access before termination of an employee, but we also propose that the DM store the information regarding the location of the data (or the location of DC) stored on the CSP (cloud log-in information, etc.) to prevent a lost or stolen DC from downloading all cloud-based data and brute-forcing the data encrypted by the symmetric key remotely. When the user authenticates itself to the DM, the DM could also establish a connection between the DC and the CSP. In this way, the end user should have little to no information about accessing the encrypted data in the cloud without prior authentication to the DM. In any case, we insist that the encryption scheme for DK contain a large key size to minimize the likelihood of a successful brute-force attack. Regardless, the malicious insider could still make a copy of the unencrypted data that may be stored in the RAM while having access to it, so appropriate secure coding as well as managerial controls are necessary.

6. LIMITATIONS

With any cloud-based encryption scheme like this, simultaneous editing of the same files would be difficult since we are not allowing the CSP to decrypt any of the contents of the files. However, we might enforce that no more than one person at a time could write to the contents of a file. For example, in a hospital system, no more than one doctor or nurse should need to access a patient's records in order to write to them at the exact same time.

We have the most important assumption regarding a practical implementation of the system; the DM and the KaaSP should always be available for the communication with the DCs and the CSP. Without a connection to the DM, it is impossible to reconstruct the keys. In this paper, we do not consider unrecoverable faults at the DM and the KaaSP which contribute to permanent data loss. To avoid this situation, we recommend using a key escrow scheme or setting up secret backups periodically.

We also assume that there is no collusion between the DM and the KaaSP because the DM is a trusted component and is managed by an SME employee.

In terms of data security, there are also a few limitations to keep in mind. First of all, without a short session time (i.e., requiring the user to re-authenticate itself frequently), the security of the system is greatly reduced. Since authentication into the system means the use of the decryption key, if a malicious attacker stole the device with the user logged in, he or she may be able to find and extract the decryption key from the device. This again is not a vulnerability unique to this system. However, we suggest that access of the files is within a bigger framework that allows for easy and instantaneous return to the work environment upon re-authentication.

Similarly, we have to assume that the device (i.e., DC) is re-authenticated regularly. The organization could specify a time limit upon registration to the DM for how long the key may exist locally on the DC to improve the usability of the system. After authentication and reconstruction of SK, the device will maintain DK locally for this time period (e.g., 12 hours). Once expired, the device will save and encrypt the data back to the state of $E_{DK}(D)$ and remove SK. The user will then need to re-authenticate in order to reconstruct SK again.

7. IMPLICATIONS

Encryption, key management, and access controls are undeniably complex and difficult processes to implement in information technology security, but compliance to rigorous national and international audit standards is even more difficult. We hope that a comprehensive framework such as this KaaSP infrastructure will be able to mitigate a large portion of the workload for SMEs looking to use cloud-based services, especially regarding ISO compliance audits.

The ISO 27001 standard has specific sections dedicated to many different forms of access control, including those for the different networks, operating systems, and user privileges etc. In the final version of our system, the KaaSP will be able to record the unique user login time, duration, and location, and logging of such activities will be available for the SMEs.

ISO 27001/2 section A.9: Access Control is a very broad standard that spans from policies to access right privilege procedures to password management while section A.10: Cryptography requires key management and encryption policies [5]. We hope that our proposed mechanism will help secure ISO certification in all areas of sections 9 and 10 from ISO 27001/2, including policies, as every organization could tailor their policies based on their needs, and our system would support that flexibility.

8. FUTURE RESEARCH

There are many subtle options that are available to the enterprise in determining how to best maintain and operate their DM, especially when regarding the backing up of employee login key information (which could also potentially be stored in the cloud). This topic on efficient and secure operation of DM warrants its own research.

We also plan to do an application of this research on a program that will access popular cloud systems currently available on the market to test the speed and practicality of this system.

9. CONCLUSION

Good security is built into the system, not added afterward. Cloud computing is very powerful, but it is still very young. This means that the development of cloud computing will face many problems. Our proposed mechanism can fit into many other proposed and existing frameworks in order to alleviate some security concerns. The Keying as a Service model has potential to operate alongside Software, Platform, Infrastructure, and other cloud services to provide more auditability of confidentiality through cryptographic means.

10. APPENDIX OF ABBREVIATIONS

ABBR.	Description
$E_K(X)$	Encryption of X by E_K
CSP	Cloud Service Provider
KaaSP	Keying as a Service Provider
DC	Domain Client
DM	Domain Manager
K	Symmetric Key
□	XOR
S	Secret generated by AONT (here, an XOR operation)
s_x	Randomly generated key part
AONT	All-Or-Nothing Transform
SMEs	Small and Medium Enterprises
ECC	Elliptic Curve Cryptography
RSA	Rivest, Shamir and Adleman
AES	Advanced Encryption Standard

11. REFERENCES

[1] Rivest, R. 1997. All-or-nothing encryption and the package transform. In *Proceedings of Fast Software Encryption* (Haifa, Israel, Jan. 20-22, 1997). Springer Berlin Heidelberg, Berlin Germany, 210-218. DOI=http://dx.doi.org/10.1007%2FBFb0052348.

[2] NSA. 2009. The Case for Elliptic Curve Cryptography. *National Security Agency: Central Security Service*. URL=https://www.nsa.gov/business/programs/elliptic_curve.shtml.

[3] GnuPG. 2014. URL=https://gnupg.org/index.html.

[4] Pearce, N. 2014. Can cloud computing overcome its crisis of confidence? *The Guardian. URL*=http://www.theguardian.com/media-network/media-network-blog/2014/feb/18/cloud-computing-nsa-privacy-breaches-crisis-confidence.

[5] International Organization for Standardization and International Electro-technical Commission. 2013. Information technology - security techniques - information security management systems - requirements (ISO/IEC 27001:2013). ISO/IEC, Berlin, Germany.

[6] Song, Y., Kim H., and Mohaisen, A., 2014. A private walk in the clouds: Using end-to-end encryption between cloud applications in a personal domain. In *11th International Conference on Trust, Privacy & Security in Digital Business* (Munich, Germany, 2014). TrustBus. URL=http://seclab.skku.edu/wp-content/uploads/2014/07/EnCloud.pdf.

[7] Hei, X. and Lin, S. 2014. Multi-part file encryption for electronic health records cloud. In *Proceedings of the 4th ACM MobiHoc workshop on Pervasive wireless healthcare* (Philadelphia, USA, Aug. 11-14, 2014). MobileHealth '14. ACM, New York, NY, 31-36. DOI=http://dx.doi.org/10.1145/2633651.2637473.

[8] Ghebghoub, Y., Boussaid, O., and Oukid, S. 2014. Security model based encryption to protect data on cloud. In *Cloud Networking, 2013 IEEE 2nd International Conference* (San Francisco, USA, Nov. 11-13, 2013). CloudNet. IEEE, 185-189. DOI=10.1109/CloudNet.2013.6710575.

[9] Zhao, F., Li, C., and Liu, C. 2014. A cloud computing security solution based on fully homomorphic encryption. In *Advanced Communication Technology, 2014 16th International Conference* (Pyeongchang, South Korea, Feb. 16-19, 2014). ICACT. IEEE, 485-488. DOI=10.1109/ICACT.2014.6779008.

[10] Moghaddam, F., Karimi, O., and Alrashdan, M. 2014. A comparative study of applying real-time encryption in cloud computing environments. In *Proceedings of the International Conference on Information Systems and Design of Communication* (Lisboa, Portugal, May 16-17, 2014). ISDOC '14. ACM, New York, NY, 50-55. DOI=http://dx.doi.org/10.1145/2618168.2618176.

[11] Bos, J, and Kaihara, M. 2010. PlayStation 3 computing breaks 2^60 barrier 112-bit prime ECDLP solved. *Ecole Polytechnique Fédérale de Lausanne.* URL=http://lacal.epfl.ch/112bit_prime.

[12] Shamir, A. 1979. How to share a secret. *Communications of the. ACM* 22 11. (New York, USA Nov. 1979), 612-613. DOI=http://doi.acm.org/10.1145/359168.359176

12. ACKNOWLEDGEMENTS

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (No. 2014R1A1A1003707) and was also funded in part by the ICT R&D program (2014-044-072-003 , `Development of Cyber Quarantine System using SDN Techniques') of MSIP/IITP.