http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (8): 1598-1606, 2010
ISSN 1812-5638

A Hierarchical Access Control Scheme in Cloud using HHECC

Yingjie Xia, Li Kuang and Mingzhe Zhu
Hangzhou Institute of Service Engineering, Center for Service Engineering,
Hangzhou Normal University, Wenyi Road 222, 310028 Hangzhou, Zhejiang, People's Republic of China

Abstract: This study proposes a novel access control scheme which is implemented based on a Hierarchical and Hybrid Elliptic Curve Cryptography (HHECC) to fulfill the efficiency and security requirements in Cloud. Cloud is a state-of-the-art technology to build a storage platform for data backup, resource sharing, file synchronization, etc. Since accessed by various users, the Cloud storage services should enhance its security facility for access control, according to the confidential protocol and privacy policy. To satisfy these requirements, we present the hierarchy feature of the scheme is achieved by an one-way hash function on the level key, merged by the hybrid feature which denotes the combination of one-way hash, Advanced Encryption Standard (AES) and ECC for the fine grained access control. The theoretical security proof of the whole scheme is figured out through logic induction with the basis on the security of ECC. The simulation experiments' results show that in comparison with RSA, HHECC facilitates to achieve much securer performance for hierarchical access control with higher efficiency for the encryption and decryption of Cloud data.

Key words: Elliptic curve crytopgraphy, access control, one-way hash, advance encryption standard, relationship graph

INTRODUCTION

Cloud is a new platform for massive storage and high-throughput computation nowadays (Vouk, 2008). As one of its main applications, the Cloud storage is used to build data center which provides various services for data backup, file synchronization and resource sharing (Ghemawat *et al*., 2003). Various kinds of users can be hierarchized to access different data in cloud through these services and there may be many complicated dependency relations among the users. In order to meet the confidential protocol and privacy policy, the cloud storage services should be enhanced with a secure access control scheme and encrypt plain data into ciphertext (Armbrust *et al*., 2010). Different users are provided with different access privileges on encrypted data. Generally speaking, authorized users include the data owner and the users with whom the owner is willing to share (Hu *et al*., 2009). The hierarchical users and their access privilege assignments comprise the hierarchical access control scheme, which is in charge of managing Cloud data utilization and security.

Access control is a fundamental mechanism to enable the authorization of access privileges on data or other resources. It can be used widely, especially in computer science, electronics and automation. Tolone *et al*. (2005) use the access control scheme in collaborative system to balance the competing goals of collaboration and security. Park *et al*. (2001) implement a RBAC model to achieve the management of enterprise-wide Web servers in large-scale network. Traditional access control models can be categorized as Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC) (Ferraiolo and Kuhn, 1992). They are implemented by several approaches, such as access matrix, Access Control Lists (ACL) and capability-based approach, with different advantages and disadvantages to various types of data accessed (Sandhu and Samarati, 1994). There are also many researchers trying to add new features to access control scheme in order to improve its performance for new infrastructures, such as access control on P2P (Tran *et al*., 2005), grid (Lang *et al*., 2006) and cloud (Hu *et al*., 2009). In most high throughput platforms, the applications are challenged on the efficiency. Therefore, in our Cloud storage service, we are working on an efficient access control scheme to regularize user privileges.

Another point focused on in this paper is the encryption. There are many algorithms used to encrypt Cloud data for access control, such as RSA (Rivest *et al*., 1978), Advanced Encryption Standard (AES) (Di Natale *et al*., 2009), hash (Chin, 1994), etc. They can be adopted with different trade-offs in security and efficiency. However, most implementations separate

Corresponding Author: Yingjie Xia, Hangzhou Institute of Service Engineering, Center for Service Engineering,
Hangzhou Normal University, Wenyi Road 222, 310028 Hangzhou, Zhejiang, People's Republic of China

access control and data encryption as two independent components, which may cause redundant interactive operations between them. Therefore, in this study, we propose a Hierarchical and Hybrid Elliptic Curve Cryptography (HHECC) scheme to encrypt Cloud data for user access control. The hierarchy feature of the scheme is achieved by one-way hash function on the key for data decryption. Boneh *et al*. (2005) designed a Hierarchical Identity Based Encryption (HIBE) based on a bilinear map computation without regards to hierarchy depth. However, HIBE algorithm is too complex to be used in Cloud because it is not efficient for the massive data access and high throughput computation. A lightweight scheme with the features of hierarchy and high security is preferred for access control in Cloud. The hybrid feature denotes the combination of hash, AES and Elliptic Curve Cryptography (ECC) (Koblitz *et al*., 2000) for a fine grained control of user access. The plain data in Cloud are encrypted by ECC which is a public-key encryption algorithm with high performance in security. The HHECC-based access control scheme is implemented as part of cloud storage service. It allows authorized users to access the corresponding data in Cloud. The simulation experiments show that in comparison with other schemes, HHECC-based scheme can achieve better performance in both security and efficiency of Cloud data access.

DESIGN OF HHECC-BASED HIERARCHICAL ACCESS CONTROL IN CLOUD

Access control in cloud: In the architecture of storage Cloud, besides the storage facility, there are other components which affect the quality of services. For instance, data recovery, power management and system cooling are as important as the Map/Reduce component when building a data center in Cloud. A four-phase Cloud Storage Maturity Model (Lesem, 2010) is proposed based on the actual storage adoption process which changes over time. Beginning from the second phase, the access option becomes a necessary component in the model. And the security component such as data encryption is also integrated in the model, especially for the higher phases.

Figure 1 shows the component model of a common data center based on storage C loud. The storage facility is the kernel component of the architecture and other components facilitate the kernel. In particular, the access control component is an indispensable part for the applications of data center in government and

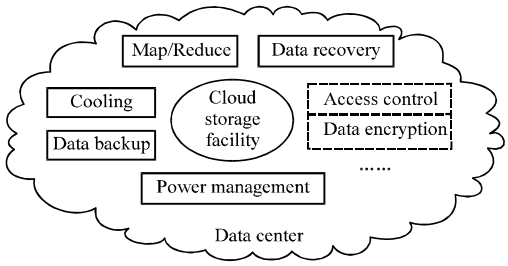

Fig. 1: Component model of a data center based on storage cloud

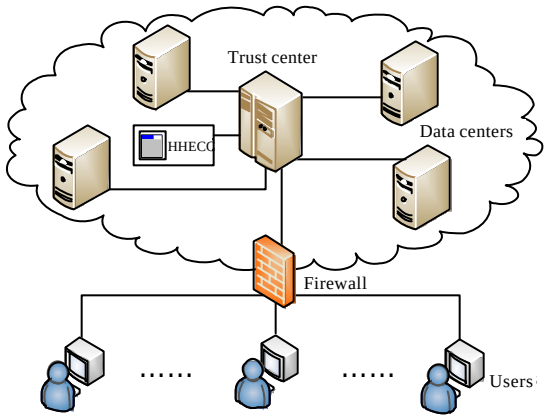

Fig. 2: Architecture of HHECC-enabled cloud

business. The data encryption component is also important for the customers with high requirements on confidentiality and security. In common architecture of data center, these two components are implemented independently. However, in our design of hierarchical access control in Cloud, we try to combine the two components. This combination is implemented through HHECC which achieves a hierarchical encryption on Cloud data and different access privileges assignments to users.

HHECC: The HHECC is an algorithm designed to encrypt Cloud data for hierarchical access control. It is a hybrid method to add hierarchical encryption to ECC which is an efficient cryptography for the cloud storage with high throughput. In Cloud, we deploy a trust center where HHECC is executed. The architecture of HHECC-enabled Cloud is shown in Fig. 2. The HHECC algorithm mainly consists of three parts, ECC, hierarchical encryption and hybrid encryption. Their designs are specified respectively as follows.

ECC: The ECC is an approach to public-key cryptography based on the intractability of finding discrete logarithm of

a random elliptic curve element with respect to an infeasible publicly-known base point. The cutting-edge research of ECC has been leveraged in some applications (Martinez *et al*., 2009; Gura *et al*., 2004) to implement their security facilities with a smaller key size and same security level in comparison with other public-key cryptographies, such as RSA.

For the aims of convenience and efficiency in current cryptographies, an elliptic curve is a plane curve which satisfies the Eq. 1:

$$y^2 = x^3+ax+b \quad (1)$$

$$4a^3+27b^2 \neq 0 \quad (2)$$

The Eq. 2 is a restriction for a and b to avoid an odd elliptic curve. In HHECC, we use the prime curve over Z_{pr} defined on a big prime number pr which bounds a finite field of discrete points from 0 to pr-1. One point is chosen as the base point G whose order is n in Z_{pr}. Therefore, all the main parameters (pr, a, b, G, n) will be specified in the implementation of ECC.

The ECC encryption is based on the add operation of elliptic curve points in Z_{pr}. Given two points on curve P and Q with their respective coordinates (x_P, y_P) and (x_Q, y_Q), the slope of their linked line λ is:

$$\frac{y_Q - y_P}{x_Q - x_P}$$

and the addition of the two points results in another point R which is defined as the X-axis mirror point of the third interaction point of the linked line and curve. R = P+Q can be calculated as below.

$$x_R = (\lambda^2 - x_P - x_Q) \bmod pr \quad (3)$$

$$y_R = (\lambda(x_P - x_R) - y_P) \bmod pr \quad (4)$$

A pair of private key d and public key PK for ECC are generated firstly, with dG = PK. d is a big integer hardly to be figured out by brute force. Then a fitting function P_M = Fit(M) is used to fit the plain data M to a point P_M modulo pr on the discrete elliptic curve. Finally, by choosing a random $k \in [1,n-1]$, P_M is encrypted into a ciphertext C_m = (kG, P_M+kPK). In decryption, the plain data M can be calculated through $Fit^{-1}(P_m+kQ-dkG)$.

HIERARCHICAL ENCRYPTION

Hierarchical encryption is an important mechanism associated with ECC in HHECC. It assigns the appropriate data access privileges to hierarchical users. The Cloud data which are encrypted by one user can be decrypted by itself and its authorized users. The hierarchical encryption used in the scheme to implement a hierarchical access control is one of key features of HHECC.

Hierarchical encryption is mainly implemented by a one-way hash function, which is a hash mapping executed in one way but not reversely. The Cloud users are classified into several hierarchies, $h_1,h_2,...,h_n$. According to the hierarchy they belong to, each of them is assigned with a level key. h_1 denotes the highest hierarchy and K_{h_1} is the level key of users with the highest privilege. The level key is used to encrypt the private key of ECC by AES cryptography.

Supposing the user in h_i is superior of user in h_j, which means i<j, level key K_{h_j} of h_j can be calculated from K_{h_i} of h_i through one-way hash function:

$$K_{h_j} = \underbrace{H(H(...H}_{j-i}(K_{h_i})...)) \quad (5)$$

Since, the hash function is one way, through j-1 times of function calls, it is straightforward to get K_{h_j} from K_{h_i}, while undoable inversely. For the superior to access the data encrypted by its juniors, it can use its own level key to get the level keys of its juniors first and then goes on decryption to obtain the private key to deal with ciphertext. All the encryption and decryption methods are combined in a hybrid model as stated below.

HYBRID ENCRYPTION

The ECC, one-way hash and AES are combined into a hybrid encryption model in HHECC. The model can achieve a fine grained access control for hierarchical users. With the implementation of hierarchical encryption on ECC private key, Cloud data can be encrypted into different levels by ECC public key. Users use one-way hash to get the required level key first and then combine the level key and user ID into an AES key which can be used in decryption to get ECC private key. Finally, this private key is levered to decrypt the related ciphertext in Cloud. The hybrid encryption executes in the trust center of Cloud and its specific working process is shown in Fig. 3.

The hybrid encryption model is mainly responsible for the management of one-way hash, AES and ECC in the trust center. By using the hierarchical encryption, we can represent the hash process as:

$$\forall i, h_i < h_j \quad K_{h_i} = \text{HierarchicalEncrypt}(K_{h_i}) \tag{6}$$

where, k_{h_i} is the level key of the owner of encrypted data. f is the function to combine k_{h_j} and KUID which is a serial number maintained in the trust center standing for user identity. The ECC private key d is encrypted by AES as:

$$d_{AES} = AES(key, d) = AES(f(K_{h_i}, KUID), d) \tag{7}$$

d_{AES} which is stored in the trust center can also be decrypted by the corresponding level key and KUID via

$$d = AES^{-1}(key, d_{AES}) = AES^{-1}(f(K_{h_i}, KUID), d_{AES}) \tag{8}$$

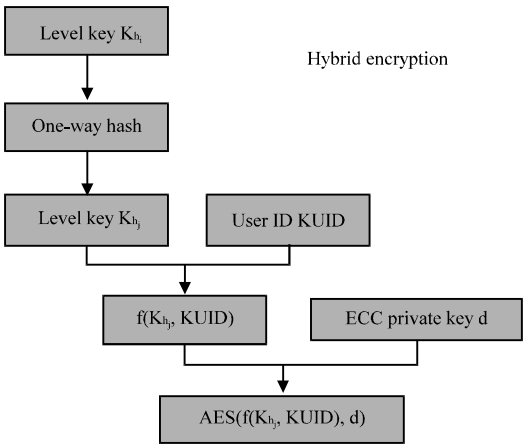

Fig. 3: Working process of hybrid encryption

d is used to decrypt the ECC encrypted data in Cloud. The whole process is implemented by hybrid encryption to authorize the user in superior hierarchy h_i or same hierarchy h_j as the owner to access the data.

IMPLEMENTATION OF HHECC-BASED HIERARCHICAL ACCESS CONTROL IN CLOUD

Hierarchical access control scheme: The HHECC provides the hierarchical access control for different users in Cloud. The trust center maintains all the user information, which is digested to generate the digital signatures. When accessing data, the user digital signature will be authenticated to locate the exact user information in trust center, such as KUID. All the users are organized in hierarchy. Their hierarchies and relationship graph are shown in Fig. 4.

The users in one hierarchy hold the same level key and two users with a directed link mean that the source user has the privilege to get KUID of the destination user. When level key and KUID are both available, the source user can access the data encrypted by destination user through the hybrid and hierarchical encryption models.

Operation protocols: The operation protocols are designed to set the system responses to various operations. There are two kinds of protocols, link operation protocols and user operation protocols.

Link add protocol: The Link Add Protocol (LAP) provides a specification to create a new directed link between a source user and a destination user therefore assigning the source user a privilege to access the data owned by the destination user. The pseudocode of LAP are shown as below.

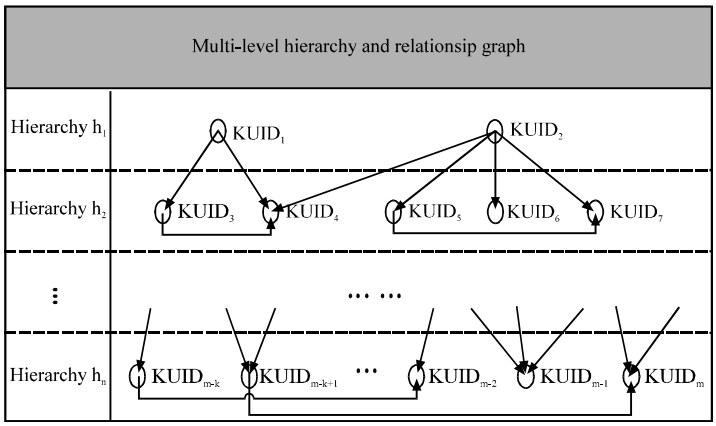

Fig. 4: User hierarchy and relationship topology

Link Add Protocol (LAP)

INPUT: i = source user, j = destination user,
G = original relationship graph
OUTPUT: G' = updated relationship graph
PROCEDURE:
if (h_j<h_i) // hierarchy of j higher than I
G' = G // link add operation failed
else
// add directed link from i to j
G' = Add_link(G, Link(i,j))
Output: G'

User i and j are the respective source and destination users of new directed link. G is the original relationship graph and G' is the updated graph with the new link. If the hierarchy of user j is higher than i, the operation fails because it is not allowed to add a link from a low hierarchy to a high one. Otherwise, the new link from i to j can be added to G and finally output G'. The added link means the source user is assigned with the privilege to locate KHID of the destination user.

Link remove protocol: When the access privilege of source user to the data owned by destination user is eliminated, the link between them should be removed correspondingly. The pseudocode of Link Remove Protocol (LRP) below shows the process to remove the link between source user i and destination user j in the original relationship graph G. The resulted relationship graph G' denotes that user i loses the access privilege to the data encrypted by user j.

Link Remove Protocol (LRP)

INPUT: i = source user, j = destination user,
G = original relationship graph
OUTPUT: G' = updated relationship graph
PROCEDURE:
// remove existed link between user i and j
G' = Remove_link(G(i, j))
Output: G'

User join protocol: When a new user joins, we firstly need to add it into a hierarchy in relationship graph and initialize its identity information, such as KUID and Level Key. Then we add corresponding links of the new user to relationship graph. The pseudocode of User Join Protocol (UJP) is as below. *l* is a set of pairs of source user and destination user for each new link. UJP uses LAP to add all the new links to original relationship graph G, therefore outputting updated relationship graph G' with new user and new links.

User Join Protocol (UJP)

INPUT: i = new user, G = original relationship graph
h_i = joined hierarchy, *l* = set of new links,
OUTPUT: G' = updated relationship graph
PROCEDURE:
// initialize KUID and Level Key of new user
Initialized(i,h_i)
G' = Add_user(G,i)

forall l_k: set of new links *l*
// add new links to the relationship graph
G' = LAP(l_k.src,l_k.dest,G')
endfor
Output: G'

User leave protocol: When a user leaves actively or passively in some cases, to forbid its operations but maintain its data, it will be isolated from the relationship graph but not removed directly. The user isolation eliminates the privileges to access the data encrypted by its linked users, while keeps the access privileges on its own data. User Leave Protocol (ULP) specifies the process of user isolation, including removing and rebuilding the corresponding links. The pseudocode of ULP is shown as below.

User Leave Protocol (ULP)

INPUT: i = leaving user, G = original relationship graph
OUTPUT: G' = updated relationship graph
PROCEDURE:
forall c_{ik}: child users of i
// remove links between i and its child users
G' = LRP(i, c_{ik}, G)
forall p_{im}: parent users of i
// rebuild links between i's parent and child users
G' = LAP(p_{im},c_{ik},G')
endfor
endfor
forall p_{im}: parent users of i
// remove links between i and its parent users
G1' = LRP(p_{im},i,G')
endfor
Output: G'

We use protocol LRP to remove all parent and child links of the leaving user i. We also use protocol LAP to rebuild links between parent and child users. The protocol ULP maintains the user ias an isolated node and tries to keep original links for associated users in the relationship graph.

Hierarchy adjust protocol: The adjustment of user hierarchy is a common operation in our scheme. There are two cases, namely, hierarchy descending and ascending. For either case, Hierarchy Adjust Protocol (HAP) should deal with the links adjustment respectively. The pseudocode of HAP is as below.

Hierarchy Adjust Protocol (HAP)

INPUT: i = adjusted user, h_j = new hierarchy,
G = original relationship graph
OUTPUT: G' = updated relationship graph
PROCEDURE:
if (h_j>h_i) // hierarchy of i descends
forall c_{ik}: child users of i
// new hierarchy of i is lower than its child user

```
    if (hj>hcik)
        G' = LRP(i,cik,G)              // remove that link
    endfor
else // hierarchy of i ascends
    forall Pim: parent users of i
    // new hierarchy of i is higher than its parent user
    if (hj<hPim)
        G' = LRP(Pim,i,g)              // remove that link
    Endfor
// update the level key of adjusted user i
Update_levelkey(i,hj)
// update the encrypted private key of adjusted user i
Update_privatekey(i)
Output: 1'
```

If new hierarchy is higher than some hierarchies of parent users of i or lower than some hierarchies of child users of i, the unsuitable links between i and these parent or child users should be removed. And all the other links should be kept. Besides the links adjustment, the new hierarchy h_j is also used to update the level key and the encrypted private key of user i.

SECURITY PROOF OF SCHEME

The security of HHECC-based access control scheme relies on three core encryption components: one-way hash function, AES and ECC. In order to clarify the security bottlenecks, we make analysis on each component and deliver security proof of the scheme.

Figure 5 shows the security components model of HHECC. There are two storage facilities in the model, trust center and data center. The trust center holds the AES encrypted private key, level keys and the relationship graph consist of KUID. The data center stores ECC encrypted data associated with data tags which correspond to the KUIDs of owners in relationship graph.

Theorem 1: If an attacker can crack HHECC-based access control scheme within feasible computational effort, it can crack ECC within less effort.

Proof: Assuming the hacking cannot get all the information related to the security components of the scheme, from the model in Figure 5, we can find that there are two inbreak points, which are encrypted private key and encrypted data. Either through cracking AES-encrypted private key or directly cracking the ECC, HHECC-based access control scheme would be broken. The cracking work to access the private key includes not only to figure out the corresponding level key and KUID, but also their combination approach. Theoretically, as a user in a lower hierarchy, it cannot reverse a one way hash to get the level keys in higher hierarchies. Even if the user has got the corresponding level key, it is still infeasible to crack without the KUID in trust center and

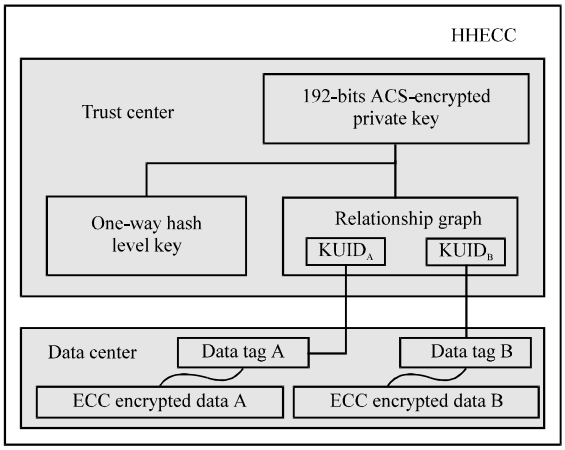

Fig. 5: Security components model of HHECC

the unique combination approach. Therefore, the hardness of AES-encrypted private key is reduced to the hardness of AES algorithm. And the overall hardness of HHECC-based access control scheme is reduced to the minimum hardness of AES and ECC.

Denote the security performance of HHECC, AES and ECC as Sec(HHECC), Sec(AES) and Sec(ECC). Based on the statement above, their relationship can be referred to as below.

$$\text{Sec(HHECC)} \geq \text{Min(Sec(AES),Sec(ECC))} \tag{9}$$

We adopt 192-bits AES encryption algorithm in our scheme since it has high security performance. Its computational effort in comparison with ECC for cryptanalysis can be evaluated as below (Stallings, 2006).

$$\text{Sec}(\text{AES}_{192\text{bits}}) \approx \text{Sec}(\text{ECC}_{384\text{bits}}) \tag{10}$$

Since, we implement ECC with the key size smaller than 384 bits in HHECC, the $\text{Sec}(\text{AES}_{192\text{bits}})$ is higher than Sec(ECC) and their relationship of security performance can be induced into:

$$\text{Sec(HHECC)} \geq \text{Sec(ECC)} \tag{11}$$

From the proof above, we can address that in our implementation the hardness of HHECC-based access control scheme is reduced to the hardness of ECC whose security performance we will analyze in Theorem 2.

Theorem 2: The cracking of ECC with a proper key size is undoable in feasible time.

Proof: It is well-known that Pollard-rho algorithm (Pollard, 1975) is one of the fastest algorithms to crack

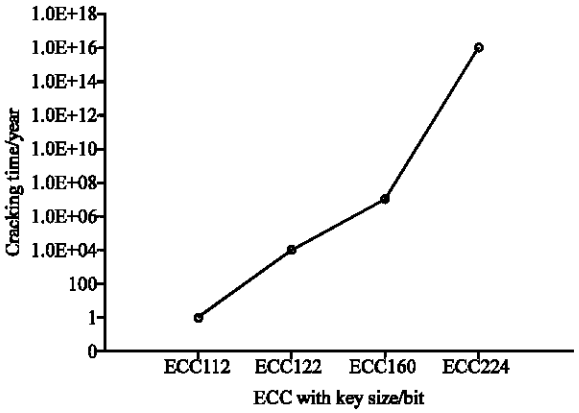

Fig. 6: ECC cracking time with different key sizes

ECC, which is efficient to solve discrete logarithm and elliptic curve logarithm. Basically, the algorithm tries to figure out two pairs of integers (a_1,b_1) and (a_2,b_2) to meet $a_1G+b_1PK = a_2G+b_2PK$. Therefore,

$$\begin{aligned} & a_1G + b_1PK = a_2G + b_2PK \\ \Rightarrow\ & (a_1 - a_2)G = (b_2 - b_1)PK \\ \Rightarrow\ & (a_1 - a_2) \equiv (b_2 - b_1)d \bmod n \\ \Rightarrow\ & d \equiv (a_1 - a_2)(b_2 - b_1)^{-1} \bmod n \end{aligned} \quad (12)$$

According to the research on Pollard-rho algorithm (Pollard, 1978), the time complexity of cracking ECC with this algorithm is $O(\sqrt{n})$ and the expected running time is roughly $\sqrt{\pi n/2}$, in which n is the order of the base point G in ECC. Therefore, it needs approximately 10^{11} MIPS years to crack 160-key-size ECC with general configurations. The ECC cracking time with different key sizes is shown in Fig. 6. As the key size increasing from 112 to 224 bits, the cracking time rapidly goes up in an exponential trend. Assuming executing the algorithm in a personal computer with 3 GHz CPU, we need around 3.3×10^7 years to break the d of 160-key-size ECC which is an unfeasible task.

RESULTS AND DISCUSSION

Experiments setup: The simulation experiments are set up on a cluster of Dawning TC4000L which has 24 nodes and 48 CPUs belonging to Zhejiang University Campus Grid. The whole project took two weeks. We deploy Hadoop package (The Apache software foundation, 2009) on the cluster to build a storage Cloud as data center. The head node of the cluster is set up as the trust center of Cloud to take charge of HHECC and others are as data center. HHECC and other simulated facilities are developed in Java in order to adapt to Hadoop. The programming environment includes JDK 1.5 and its corresponding supplementary API package of Bouncy Castle (Bouncy Castle Provider, 2007) which supports some advanced security technologies including ECC.

Table 1: Security level comparisons between HHECC and RSA

Security level	HHECC (key size/bits)	RSA (modulus size/bits)	Key size ratio
1	160	1024	1:07
2	224	2048	1:10
3	256	3072	1:12
4	384	7680	1:20

The simulation experiments test security and efficiency of HHECC-based hierarchical access control scheme. The experiment on security compares HHECC with RSA in various security levels theoretically and the experiment on efficiency tests the average time of each encryption and decryption transaction of HHECC and RSA.

Experiment on security: In security experiment, we analyze to crack HHECC and RSA by Pollard-rho algorithm (Pollard, 1975) and Number Field Sieve (NFS) (Lenstra *et al*., 1993) respectively. From the theories of both cracking algorithms, RSA is proved to be cracked in the order of sub-exponential time, while ECC is within exponential time. We define the security level which means that the time complexities for HHECC cracking and RSA cracking are similar. The four evaluated security levels of HHECC with different key sizes and RSA with different modulus sizes are listed in Table 1.

From the data in Table 1, HHECC with key size of 160 bits is as secure as RSA with modulus size of 1024 bits. They belong to Security Level 1 which is the minimum security performance among all the four levels. From Security Level 1 to Security Level 4, the securer both algorithms are, the larger their key size ratios become.

Experiment on efficiency: In efficiency experiment, we compare HHECC-based hierarchical access control scheme with a scheme of role-based access control and RSA in the average time of one transaction which involves one round process of data encryption and decryption. We test both schemes with different key sizes in each Security Level according to the four levels classified in the security experiment. Each test case executes seven times and we eliminate the maximum and minimum value which may be caused by some extreme instances, to calculate average time of the rest five values. We use MD5 as one-way hash function and AES with

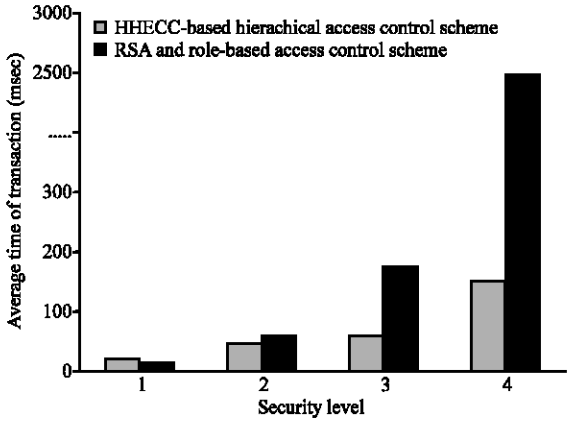

Fig. 7: ATT of two test schemes in different security levels

192 bits key size in this simulated experiment. The user who is willing to access data is set two hierarchies higher than data owner. Figure 7 illustrates Average Time of Transaction (ATT) of two test schemes in different security levels.

In the figure, we can see the bar RSA scheme increase quite faster than our HHECC scheme's and have a great gap on security level 3 and 4.

Discussion and analysis: In the experiment on security, the data in Table 1 shows that HHECC can achieve good performance on security with smaller key size, which also can be considered as high efficiency in the encryption and decryption transaction. From current researches on RSA we can know that the RSA with a 1024 key size can be considered as a safe level (Gura *et al*., 2004), so the security of HHECC can be guaranteed. And it also shows that key size of HHECC increases much more slowly than that of RSA modulus size, hence HHECC is qualified to meet the requirements of security in high-throughput cloud and more feasible than RSA in the application of cloud storage.

In the experiment on efficiency, we can find that the proposed HHECC-based hierarchical access control scheme performs more efficient than RSA and Role-based access control scheme in all Security Levels except Security Level 1 in Fig.7. This exception is caused by the combination of MD5, AES and ECC which slows down HHECC-based encryption and decryption in Security Level 1. For the rest of Security Levels, the efficiency advantage of HHECC compared with RSA takes the dominance to affect ATT and therefore the gaps between two schemes are growing larger. We can also find out that by the security level increasing the ATT of HHECC-based hierarchical access control scheme grows much more slowly than the other one, because the key size of HHECC facilitates the operations more than that of RSA.

However, we also have to admit that the scheme is not that efficient on low security level, since it has complex architecture which may be not be suitable for all data center applications. Besides the complex hybrid architecture, the difficulty of ECC algorithm is also a problem in projects. So we do not suggest the simple and small data center projects using our HHECC scheme. It is more suitable for large data center or Cloud service projects which need high security and efficiency.

CONCLUSIONS AND FUTURE WORK

In this study, we propose a novel hierarchical access control scheme based on HHECC algorithm, which integrates ECC, AES and one-way hash function to authorize hierarchical users to access encrypted Cloud data. AES encrypts private key of ECC via using the key generated by combining KUID of user and Level Key K from the hash of user hierarchy. Through HHECC users can share their data with the authorized users. The simulated experiments show that this scheme achieves a good performance on the efficiency and security of Cloud data access.

In the future work, we suggest improving the security in data transmission and key management in Cloud. We also plan to refine this scheme, such as to use multi-key method to implement multi-backup in Cloud which can support to recover data from failure more robustly. Moreover, we plan to implement some mechanisms which support to customize hierarchy based access control or fine grained identity based access control through the operations on KUID. Finally the simulated experiments will be extended to a large scale network to carry out practical tests.

ACKNOWLEDGMENTS

Authors thank a project supported by Scientific Research Fund of Zhejiang Provincial Education Department under grant number Y200907427 and Scientific Research Fund of Hangzhou Normal University under grant number HSKQ0042. We also do appreciate the helpful discussions among staff of Center for Service Engineering in Hangzhou Normal University.

REFERENCES

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph and R. Katz *et al*., 2010. A view of cloud computing. Commun. ACM., 53: 50-58.

Boneh, D., X. Boyen and E. Goh, 2005. Hierarchical identity based encryption with constant size ciphertext. Lecture Notes Comput. Sci., 3494: 440-456.

Bouncy Castle Provider, 2007. Legion of the bouncy castle java cryptography APIs. http://freshmeat.net/projects/bouncycastlecryptoapi.

Chin, A., 1994. Locality-preserving hash functions for general purpose parallel computation. Algorithmic, 12: 170-181.

Di Natale, G., M. Doulcier, M.L. Flottes and B. Rouzeyre, 2009. A reliable architecture for parallel implementations of the advanced encryption standard. J. Electr. Testing, 25: 269-278.

Ferraiolo, D. and R. Kuhn, 1992. Role-based access control. Proceedings of 15th National Computer Security Conference, (NCSC'92), Baltimore MD, pp: 554-563.

Ghemawat, S., H. Gobioff and S. Leung, 2003. The google file system. ACM SIGOPS Operat. Syst. Rev., 37: 29-43.

Gura, N., A. Patel, A. Wander, H. Eberle and C.S. Sheueling, 2004. Comparing elliptic curve cryptography and rsa on 8-bit cpus. Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, LNCS., 3156, August 2004, Springer, Berlin, Heidelberg, pp: 119-132.

Hu, L., S. Ying, X. Jia and K. Zhao, 2009. Towards an approach of semantic access control for cloud computing. Lecture Notes Comput. Sci., 5931: 145-156.

Koblitz, N., A. Menezes and S. Vanstone, 2000. The state of elliptic curve cryptography. Des. Code Cryptogr., 19: 173-193.

Lang, B., I. Foster, F. Siebenlist, R. Ananthakrishnan and T. Freeman, 2006. Attribute based access control for grid computing. http://www.mcs.anl.gov/uploads/cels/papers/P1367.pdf.

Lenstra, A.K., H.W. Lenstra, Jr., M.S. Manasse and J.M. Pollard, 1993. The number field sieve. Proceeding 22nd ACM Symposium Theory of Computing, (ACMSTC'93), Springer, Berlin, pp: 564-572.

Lesem, S., 2010. The cloud storage maturity model: A value-driven process. http://cloudstoragestrategy.com/2010/03/the-cloud-storage-maturity-model-a-value-driven-process-1.html.

Martinez, V.G., L.H. Encinas and C.S. Avila, 2009. Elliptic curve cryptography. Java platform implementations. Int. J. Inform. Technol. Security, 4: 65-72.

Park, J.S., R. Sandhu and G. Ahn, 2001. Role-based access control on the web. ACM Trans. Inform. Syst. Secur., 4: 37-71.

Pollard, M.J., 1975. A monte carlo method for factorization. BIT Numerical Math., 15: 331-334.

Pollard, J.M., 1978. Monte carlo methods of index computation (mod p). Math. Comput., 32: 918-924.

Rivest, R.L., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public-key cryptosystems. ACM Commun., 21: 120-126.

Sandhu, R.S. and P. Samarati, 1994. Access control: Principles and practice. IEEE Commun. Magazine, 32: 40-48.

Stallings, W., 2006. Cryptography and Network Security: Principles and Practice. 4th Edn., Prentice Hall, Upper Saddle River, New Jersey, ISBN: 0130914290.

The Apache Software Foundation, 2009. The hadoop distributed file system architecture. The Apache Software Foundation.

Tolone, W., G. Ahn, T. Pai and S. Hong, 2005. Access control in collaborative systems. ACM Comput. Surv., 37: 29-41.

Tran, H., M. Hitchens, V. Varadharajan and P. Watters, 2005. A trust based access control framework for P2P file-sharing systems. Proc. Ann. Int. Conf. Syst. Sci., 9: 302c-302c.

Vouk, M.A., 2008. Cloud computing-issues, research and implementations. J. Comput. Inform. Technol., 4: 235-246.