Homomorphic-Encryption-Based Separation Approach for Outsourced Data Management*

Yang Zhang and Jun-Liang Chen

State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts & Telecommunications,
Beijing 100876, China
YangZhang@bupt.edu.cn

Abstract. With the rapid application of cloud computing technologies, service and data outsourcing has become a practical and useful paradigm. In order to manage sensitive information in this outsourcing scenario, combined use of access control technologies and cryptography was proposed by many researchers. However, the rigid combination in existing approaches has difficulty in satisfying the flexible data management for diverse applications. In this paper, we advocate a separation methodology where an authorization policy is not required to be embedded into ciphertexts or keys during encrypting data, and can be linked to the ciphertexts at any time. Authorization is independently carried out as usually without involving encryption, and encryption plays a foundational mechanism without considering authorization. We propose a separation approach based on homomorphic encryption to realize outsourced data management, where an encryption procedure is separated from authorization, and dynamically integrated with authorization policy according to subjects' attributes at any time.

Keywords: Outsourced Data Service, Access Management, Homomorphic Encryption.

1 Introduction

1.1 Motivation

When cloud services become popular on the Internet, users are more and more resorting to service providers for publishing resources shared with others. Service providers are requested to realize data and service outsourcing architecture on a wide scale. For example, YouTube and MySpace are such service providers. Their basic assumption that service providers have complete access to the stored resources is not applicable for all actual scenarios such as outsourcing sensitive data. Current solutions adopt

* Supported by Project of New Generation Broadband Wireless Network under(Grant No.2010ZX03004-001, 2011ZX03002-002-01, 2012ZX03005008-001).

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 24–34, 2013.

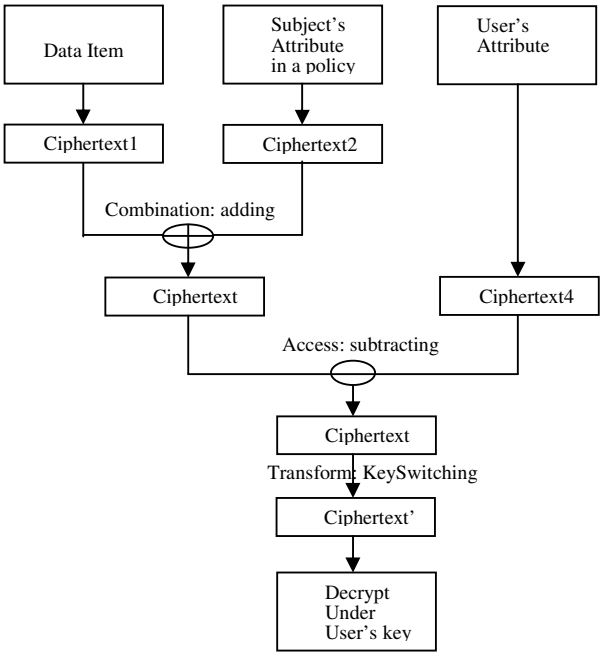

Fig. 1. Separation based on encryption homomorphism

encryption techniques instead of the legal protection offered by contracts when enforcing access control, i.e. the data owner encrypts data, sends ciphertexts to the service providers for storage, and distributes the corresponding key to authorized users [1], [2], [3], [4], [5].

In particular, since neither the data owner nor the service providers can solely assume the access control task for either efficiency or security reasons, realizing access control via the stored data themselves appears more applicable. As an example, the selective encryption technique was adopted to encrypt different data with different keys, which guarantees that legal users can retrieve the key to decrypt the encrypted resource [6], [7], [8], [9], [4], [12], [13]. However, those approaches need user to derive all the keys of authorized data items according to a public catalog of tokens, or assume that users' attributes are a set without inner structures. The complexity of authorization policies is passed to cryptographic operations, and end users directly face with it. This also results in static data management without supporting practical attribute structure, and policies are hard to be updated. Therefore, we advocate a relative separation methodology, where authorization and cryptographic operations both independently carried out, and are transparently combined for users.

In this paper, according to the separation methodology, the authorization policies are independently made and data is independently encrypted. When a policy should be applied to a data item which could have become a ciphertext, the data owner translates

the policy into another ciphertext and uses a homomorphic function to get the sum of the two ciphertexts (add ciphertexts), i.e. combining the policy with the data item. The outsourcer has the ciphertext of a user's attribute, and subtracts this ciphertext from the sum. If the user's attribute is equal to the subject's attribute in the authorization rule, the sum is transformed into the ciphertext of the data item. This cloud data management idea is illustrated in the below figure 1.

From the figure 1, we can know the owner does combination, i.e. adding ciphertexts, the outsource does subtracting and transforming, and the user transparently decrypts ciphertexts under her or his private key. If she or he has the proper attribute, the user gets a correct ciphertext, i.e. the value of *Ciphertext3 – Ciphertext4* being equal to *Ciphertext1*. The key points are designing homomorphic function and preventing homomorphic attacks. The homomorphism brings designers great power. It also provides adversaries a homomorphic attack method. The scheme designers should try to limit the capacity of adversaries.

1.2 Contributions

Contributions of our paper are as follows:

1. A separation methodology is adopted to get a flexible data management framework for cloud data services.
2. Based on the homomorphic key-switching scheme, a cloud data management solution is proposed. The data owner can authorize before, during, or after doing encryption. The complex subject attribute relationship is supported.

2 Preliminaries

2.1 Attribute-Based Authorization Policy

We adopt the attribute-based access control model.

Definition 1. Attribute Tuple. *The attribute of a subject* S *is denoted by* $s_k = (s\_attr_k, op_k, value_k)$ *and the attribute of an object* O *as* $o_n = (o\_attr_n, op_n, value_n)$, *where* s_attr *and* o_attr *are attribute names, value is the attribute value, and* op *is some attribute operator such as* $op \in \{=, <, >, \leq, \geq, in\}$. *The action attribute can be one of object's attribute. The attribute tuple is* $< s_1, s_2, \cdots, s_K >$ *or* $< o_1, o_2, \cdots, o_N >$, *where the relationship among the attributes is conjunction. S can be represented by the set of attribute tuples* $\{< s_1, s_2, \cdots, s_K >\}$, *and* O *by* $\{< o_1, o_2, \cdots, o_N >\}$.

In our paper, the op is simplified as $\{=\}$ by describing digital attributes with careful intervals.

Definition 2. Authorization Rule. *An attribute-based rule is* $rule = (< s_1, s_2, \cdots, s_k >, < o_1, o_2, \cdots, o_n >) = (SA, OA)$. *The* $j-th$ *subject attribute is written as* $rule.s_j$. *The* $j-th$ *object attribute is written as* $rule.o_j$.

An access control policy AP_i can be defined as $AP_i = \bigcup_{j=1}^{L} rule_{i,j}$. Then,

$AP_i.SA = \bigcup_{j=1}^{L} rule_{1,j}.SA$, $AP_i.OA = \bigcup_{j=1}^{L} rule_{i,j}.OA$.

2.2 Access Expression

Let Γ be an access expression representing the subject attributes required to access some data, which uses logic operators to associate the attributes. According to the authorization policy $AP.SA = rule_1.SA \cup \cdots \cup rule_l.SA$, the access expression Γ could be represented as $\Gamma = (w_{1,1} \wedge \cdots \wedge w_{1,n_1}) \vee \cdots \vee (w_{l,1} \wedge \cdots \wedge w_{1,n_l})$, where $w_{i,j} ::= "s\_attr_{i,j} = value_{i,j}"$, $1 \le i \le l$, $1 \le j \le n_l$.

Definition 3 (Satisfying an access expression Γ**).** *If a subject attribute expression* $\gamma = (w_{1,1} \wedge \cdots \wedge w_{1,n'_1}) \vee \cdots \vee (w_{l',1} \wedge \cdots \wedge w_{1,n'_{l'}})$ *satisfies* Γ, *we denote it by* $\Gamma(\gamma) = 1$, *else* $\Gamma(\gamma) = 0$. *We compute* $\Gamma(\gamma) = 1$ *as follows:*

For $(w_{i,1} \wedge \cdots \wedge w_{i,n_i}), 1 \le i \le l$ *in* Γ, *there is a* $(w_{j,1} \wedge \cdots \wedge w_{1,n'_j}), 1 \le j \le l'$ *in* γ *satisfying:*

For each $w_{i,x}, 1 \le x \le n_i$ *in* $(w_{i,1} \wedge \cdots \wedge w_{i,n_i})$, *there is a* $w_{j,x'}, 1 \le x \le n'_j$ *satisfying* $w_{j,x'} = w_{i,x}$.

We add such $(w_{i,1} \wedge \cdots \wedge w_{i,n_i})$ *into the set* Δ *which is filled with all such rules (sub-expression).*

We denote Δ *as* $\Gamma \bigcap \gamma$, *i.e.,* $\Gamma \bigcap \gamma = \Delta$. *If* Δ *is not empty, we think that* γ *and* Γ *are matching, i.e.* $\Gamma(\gamma) = 1$.

In a word, the access expression Γ for a data item represents the subject attributes in authorization rules of the data owner, and the subject attribute expression γ represents a user's attributes. These attributes have inner structures, i.e. attribute conjunction and disjunction.

3 Separate Data Management Framework

The Figure 2 demonstrates our core idea to realize separate data management for cloud services. During authorization, the data owner chooses a random value as an

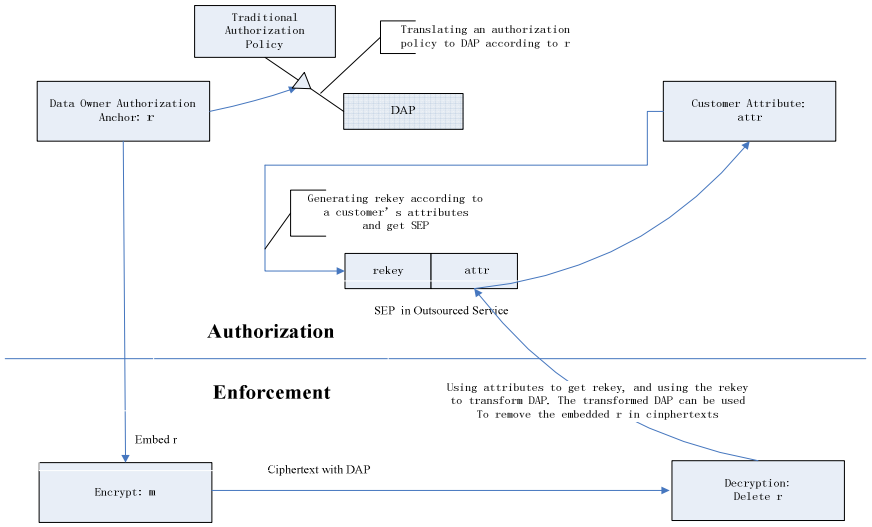

Fig. 2. Data Management Framework

authorization anchor r and translates authorization policies into data access polices (DAP) according to the anchor r. DAP can be informally considered as the ciphertexts for the subject attributes in authorization rules. The anchor r is used to randomize DAP such that even if the same authorization rule is applied to different data items, there are different DAPs which are not interchangeable. DAP can be generated before operating data encryption because it is linked to a random r. Subject encryption policies (SEP) are generated according to customers' attributes $attr$ at any time, and can also be considered as the ciphertexts for $attr$. At the same time, re-encryption keys are given to the outsourced data service who uses them to keyswitch ciphertxts, i.e. translating one ciphertext under the data owner's public key into another ciphertext under a user's public key. In our framework, each participant has only one public/private key pair to identify herself. During runtime, the encryption scheme allows for embedding the anchor r in the ciphertext. The outsourced data service makes matching between SEP and DAP, and gets one matched DAP if the requesting user has been granted the privilege. The matched DAP is keyswitched. The keyswitched DAP includes the anchor r. The keyswitched DAP can be used to remove the anchor r in the data ciphertext. The user recovers the data by her or his private key.

There are two difficulties. One is how to embed and remove the anchor, where the anchor is secret. The other is how to decrypt the ciphertext under the data owner's key, where we do not hope to carry out complex key distribution schemes to issue a series of keys to the customer. In this paper, we adopt a homomorphic encryption scheme to address the first issue. We design a switching key approach to address the second issue. That is to say, a ciphertext under the data owner's keys can be switched into a ciphertext under customers' keys. Our solution has the following characteristics:

1) The attributes of a subject can have inner structures, that is to say, there are logic 'AND' and 'OR' relations among them.
2) The authorization policies can be generated or modified after the ciphertexts have been computed. It can also be generated before the ciphertexts are produced.

4 Fundamental Encryption Scheme

We adopt the fully homomorphic encryption scheme in [10] which is CPA-secure based on the RLWE (Ring Learning with Errors) assumption. We rewrite the scheme in our words (some details can be found in [10]). According to the encryption scheme, we modify a little the key switching scheme, where the customer's private key need not be disclosed to the data owner during generating switching keys. If a reader is familiar with the encryption scheme in [10], she or he could skip this section.

4.1 Homomorphic Encryption

The following homomorphic encryption scheme only supports multiplication one time, which is a simplified version of the one in [10]. A prime $t < q$ defines the message space $R_t = Z_t[X]/f(x)$, the ring of integer polynomials modulo $f(x)$ and t. We can adopt the message encoding algorithm used in [11] to optimize performance.

— $FHE.Setup(1^{\lambda})$: It takes as input the security parameter λ. It chooses appropriately the dimension d, which is a power of 2, the modulus q which is a prime such that $q = 1 (mod\ 2d)$ and $R = Z_q[X]/(x^d + 1)$, a prime t which defines the message space, an error distribution χ over R which depends sub-logarithmically on q ($d = \Omega(\lambda \cdot log(q/B))$), to ensure that the scheme is based on a RLWE instance that achieves 2^{λ} security against known attacks. Let $N = \log q$ and let $params = (q, d, \chi)$.

— $FHE.SecretKeyGen(params)$: It draws $s' \leftarrow \chi$. Set $sk = s \leftarrow (1, s') \in R^2$.

— $FHE.PublicKeyGen(params, sk)$: It takes as input a secret key $sk = s \leftarrow (1, s')$, which $s[0] = 1$ and $s' \in R$ and the parameters $params$. It generates matrix $A' \leftarrow R^{N \times 1}$ where a vector $e \leftarrow \chi^N$ and set $b \leftarrow A's' + te$. Sets A to be 2-column matrix consisting of b followed by the columns of $-A'$. (Obs erve: $A \cdot s = te$). Set the public key $pk = A$.

— *FHE.Enc*$(params, pk, m)$: To encrypt a message $m \in R_t$, it sets $\tilde{m} = (m,0) \in R^2$, samples $r \leftarrow R_t^N$ and outputs the ciphertext $c \leftarrow \tilde{m} + A^T r \in R^2$.

— *FHE.Dec*$(params , sk, c)$: It outputs $m \leftarrow [[< c, s >]_q]_t$ or $m \leftarrow [[< c, s \otimes s >]_q]_t$ if the length of c is longer than N , where $<,>$ means the dot product of two vectors, and $[]_q$ means *mod q*.

— *FHE.Add*$(params , pk , c_1 , c_2)$: It takes two ciphertexts encrypted under the same key. It sets $c_3 = c_1 + c_2$.

— *FHE.Mult*$(params , pk , c_1 , c_2)$: It takes two ciphertexts encrypted under the same key. The new ciphertext, under the secret key $\bar{s} = s \otimes s$, is the coefficient vector c_3 of the linear equation $L^{long}_{c_1,c_2}(x \otimes x)$ where $L^{long}_{c_1,c_2}(x \otimes x) = < c_1, x > \cdot < c_1, x >$, $x \otimes x$ is the tensoring of x with itself. $c_3 = (c_1[0]c_2[0], c_1[0]c_2[1] + c_1[1]c_2[0], c_1[1]c_2[1])$.

Decryption works correctly because:

$$[[< c, s >]_q]_t = [[(\tilde{m}^T + r^T A) \cdot s]_q]_t$$

$$= [[m + t \cdot r^T e]_q]_t$$

$$= [m + t r^T e]_t$$

$$= m$$

The ciphertexts produced by *FHE.Enc* contains two ring elements. Homomorphic addition is done by simple component-wise addition of two ciphertexts. For homomorphic multiplication, letting each ciphertext being the coefficients of the polynomial of symbolic variable, we can symbolically (treating the symbolic variable as an unknown one) open the parenthesis in $L^{long}_{c_1,c_2}(x \otimes x)$ to compute the product. The ciphertext product is decrypted under the key $\bar{s} = s \otimes s$, i.e. $m_1 \times m_2 \leftarrow [[< c_3, s \otimes s >]_q]_t$.

4.2 Proxy Key Switching Scheme

Key switching consists of two procedures: first, a procedure takes as input a customer's public key and a data owner's private key, and outputs some switching keys that enables the switching; and second, a procedure takes the switching keys and a

ciphertext encrypted under the data owner's public key, and outputs a new ciphertext that encrypts the same message under the customer's public key. Although it is inspired by the key switching algorithm in [10], our key switching scheme does not require the customer to provide its private key, such that ours can be used in outsourcing scenarios. When the data owner gets the customer's public key, it chooses a secret random matrix B_j which is used to blind the public key of customers. The customer cannot recover the B_j even if it gets the blinded public key. The data owner then embeds its private key in the blinded key. When the embedded public key is used to re-encrypt a ciphertext, the decryption under the embedded private key and encryption under the customer's public key take place at the same time. According to the $FHE.PublicKeyGen$ algorithm, the random matrix does not impair the customer's decryption capability. The tensor of data owner's private key can be used when switching the product of ciphertexts, where no modulus switching implies that the ciphertext multiplication is supported only once.

The following two functions were introduced in [10].

(1) $BitDecomp(x)$: $x \in R^n$. $x = \sum_{j \in [0,\cdots,\log q]} 2^j \mu_j$, $\mu_j \in R_2^n$. Let $L = n \log q$, output $(\mu_1, \cdots, \mu_{\log q}) \in R_2^L$

(2) $Powersof2(x)$: $x \in R^n$. Output $(x, 2x, \cdots, 2^{\log q} x) \in R_q^L$

If it is known a priori that x has coefficients in $[0, B]$ for $B << q$, then $BitDecomp$ can be optimized in the obvious way to output a shorter bit-decomposition in R_2^{logB}.

Observe that $BitDecomp$ and $Powersof2$ do not affect the dot product, in the following sense (Referring to [10]):

For vectors c and s of equal length, we have

$$< BitDecomp(c), Powersof2(s) >=< c, s > mod q$$

The proxy key switching scheme is as follows:

$Proxy.KeyGen(i, j)$: i generates key (sk_i, pk_i), j generates key (sk_i, pk_i), $pk_j = (b_j, -A_j') \in R^{N \times 2}$.

$Proxy.ReKeyGen(sk_i, pk_j')$: i randomly chooses $e' \leftarrow \chi^{3N}$, $B_j \in R_2^{3N \times N}$, computes $pk_j' = (b_j', A_j'') = (B_j \cdot b_j, B_j \cdot (-A_j'))$, $rekey_{i \to j} = (b_j' + Powersof2(sk_i \otimes sk_i) + t \cdot e', A_j'')$

$Proxy.Enc(pk_i, m)$: $c = FHE.Enc(pk_i, m)$.

$Proxy.SwitchKey(rekey, c)$: $c' = BitDecomp(c) \cdot rekey_{i \to j}$.

$Proxy.Dec(sk_j, c')$: $m = FHE.Dec(sk_j, c')$.

The key switching procedure preserves the correctness of decryption under the new key because

$$< c', sk_j >= BitDecomp\ (c)^T \cdot rekey_{i \to j} \cdot s_j$$

$$= BitDecomp\ (c)^T \cdot ((B_j \cdot e + e')t + Powersof\ 2(s_i \otimes s_i))$$

$$= t \cdot < BitDecomp\ (c), ((B_j \cdot e + e') > + < BitDecomp\ (c)\ , Powers\ of\ 2(s_i \otimes s_i) >$$

$$= t \cdot < BitDecomp\ (c), ((B_j \cdot e + e') > + < c, s_i \otimes s_i >$$

$$= t \cdot < BitDecomp\ (c), ((B_j \cdot e + e') > + < c, s_i > .$$

5 Separation Construction

In our construction, each participant has one public/private key pair to identify herself. We adopt an anchor-embedding approach to translate the data owner i's authorization polices into DAPs.

All attributes in one conjunction of the access expression are bound together through the hash function and a random, where the random assures that even if two conjunctions are equal, their DAPs are different. If a user's attributes are the same as the subject's attributes in the authorization rules (access expression), the user can remove the embedded attributes, the random, and the anchor from the data ciphertext to recover data. Each attribute of the user is independently translated into SEP such that SEPs can compose as needed. The binding function through hash (such as ho_x) is used to prevent homomorhic attacks. For example, faking a attribute ciphertxet through homomorhic adding and multiplication of attribute ciphertexts is hard. We then define the detail construction.

The attacks to the construction include (1) the outsourced data service uses its stored re-encryption keys, DAPs and all users' SEPs to recover the secret data; (2) users use attributes and their key-pairs to access to unauthorized data; (3) the corrupted data service and users collude to disclose secret data. The homomorphic attack is the primary threat.

For the first kind of attack, the data service can use stored re-encryption keys to switch any DAP to the ciphertext under any user's public key, and does subtraction to remove the authorization anchor and embedded attributes from the data ciphertext. Because the data service does not have users' private keys, it cannot remove the binder ho_x. The binder ho_x is computed by the one-way function (hash function) on attributes and the random r_x. If the random r_x is unknown, the probability to correctly guess ho_x is negligible. The ciphertext for ho_x and the attribute conjunction cannot be faked by homomorphic attacks because ho_x is one-way. Thus, the data service cannot get the correct ciphertext for the data item. This also prevents the curious behavior of the data service from leaking secrets, that is to say, the data service is not corrupted, but it does some unscheduled actions such as sending any ciphertext to any user.

For the second kind of attack, the DAPs are ciphertexts encrypted under the data owner's public key and the SEPs are ciphertexts encrypted under users' public key such that the homomorphic function cannot operate on them. Users have not re-encryption keys to switch the key of DAPs. Because ϖ_i is secret, $\varpi_i w_{y,x}$ cannot be computed. Users' attributes cannot be interwoven either, i.e. inserting one attribute in one conjunction into another conjunction for binding each attribute in one conjunction. Users only by themselves cannot disclose the data included in ciphertexts.

For the third kind of attack, we cannot prevent. So we assume the outsourced data service is half honest. That is to say, the data service does not collude with users, but it may try to get some secret by itself or send any ciphertext to any user.

6 Conclusions

In this paper, we address the problem of how to realize separate cloud data management in this outsourcing scenarios to manage the sensitive data of owners. The homomorphic encryption scheme is adopted as a foundation for our framework, and the homomorphic attack is also discussed. Based on the former, the relative separation could be achieved with combining encryption and authorization. The solution also realizes simple key management and the capacity to compose attributes with inner structure. How to prevent the collusion of the data service and users is left open, which is also our future research target. Therefore, our solution will provide strong building blocks for the design and implementation of manage the sensitive information in cloud computing.

References

1. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM Trans. on Information and System Security 8(1), 119–152 (2005)
2. Hacigumus, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE 2002, pp. 29–39. IEEE Computer Society, Washington (2002)
3. Hacigumus, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of ACM SIGMOD 2002, pp. 216–227. ACM, New York (2002)
4. De Capitani di Vimercati, S., Foresti, S., Jajodia, S.: Preserving Confidentiality of Security Policies in Data Outsourcing. In: Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society, pp. 75–84 (2008)
5. Samarati, P., de Capitani di Vimercati, S.: Access Control: Policies, Models, and Mechanisms. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 137–196. Springer, Heidelberg (2001)
6. Damiani, E., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective Data Encryption in Outsourced Dynamic Environments. Electronic Notes in Theoretical Computer Science, 127–142 (2007)

7. Damiani, E., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Metadata Management in Outsourced Encrypted Databases. In: Jonker, W., Petković, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 16–32. Springer, Heidelberg (2005)
8. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: management of access control evolution on outsourced data. In: Proc. of the 33rd VLDB Conference, Vienna, Austria, pp. 123–134 (September 2007)
9. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: A data outsourcing architecture combining cryptography and access control. In: Proc. of the 1st Computer Security Architecture Workshop, Fairfax, VA, pp. 63–69 (November 2007)
10. Gentry, C.: Fully Homomorphic Encryption without Bootstrapping (2011), http://eprint.iacr.org
11. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can Homomorphic Encryption be Practical, http://eprint.iacr.org/2011/133.pdf
12. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption Policies for Regulating Access to Outsourced Data. ACM Transactions on Database Systems, 1–45 (2010)
13. Yu, S.C., Wang, C., Ren, K., Lou, W.J.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: IEEE INFOCOM (2010)