2010 34th Annual IEEE Computer Software and Applications Conference Workshops

Toward cloud-based key management for outsourced databases

Nadia Bennani
LIRIS,INSA-LYON
Lyon, France
Email: nadia.bennani@liris.cnrs.fr

Ernesto Damiani
Universitá degli Studi di Milano
Milano, Italy
Email: ernesto.damiani@unimi.it

Stelvio Cimato
Universitá degli Studi di Milano
Milano, Italy
Email: stelvio.cimato@unimi.it

***Abstract*—A major drawback of implementing *Database-as-a-Service* (DaaS) on untrusted servers is the complexity of key management required for handling revocation. In this paper we put forward the idea of using the cloud for decoupling the management of local, user-specific encryption keys from the one of role-specific protection keys, obtaining simple key management and revocation schemes.**

***Index Terms*—Cloud, secret sharing, DB externalisation.**

I. Introduction

In the "database-as-a-service" (DaaS) model, a *data owner* calls in a third party (*the service provider*) to host her database [11], and provide clients with seamless access to data. Data owners can thus concentrate on their core competencies while expecting outsourced databases to be managed by the best experts using innovative solutions at low costs. This approach, it is hoped, leads to an increase in productivity as well as cost savings. Nevertheless, outsourcing databases poses several security research questions related to data confidentiality, privacy, authentication and data integrity. Foremost among them is the issue of data privacy. In some applications, data owners view their data as valuable assets to be revealed to authorized clients only, and prefer to avoid disclosing them to the service provider. Database encryption has been proposed since long as a solution to implement the DaaS model even when the service provider is untrusted: the database is encrypted by the data owner, who also hands out decryption keys, allowing only authorized persons to access the plaintext. Some seminal papers [12], [1], [11] have proposed to store additional indexing information with the encrypted database. These indexes are then used by the host DBMS to compute answers to the users queries over the encrypted data, i.e. without disclosing neither the query parameters nor its results. Figure 1 describes the mechanism for querying an encrypted database following this approach.

First, the user sends the query to the owner who maintains metadata needed to translate it to the appropriate representation on the server (1). Then, the transformed query is executed on the encrypted database at the server side (2). Once executed, the results are sent encrypted to the owner who decrypts them and filters out those tuples not satisfying the users assigned rights (3). Finally, the results are sent to the user in plaintext (4). However, this technique is clearly impractical [1], [12], [6], [14] as it requires a continuous online presence of the data owner to enforce access control. To remove this limitation, other query processing approaches [15], [5], [8], [14] have been proposed, based on shifting access policy enforcement at the client-side. In this case, the data owner's task is to identify the set of keys necessary for each user (or group) to access the data she is (they are) authorized to view. The main advantage of this idea is that the data owner is not directly involved when the user queries the encrypted database. All the proposed solutions use the *key derivation principle* [2] to reduce the number of delivered keys. However, handing out the database encryption keys to users creates a problem when a user's rights are revoked, as key revocation requires full database re-encryption.

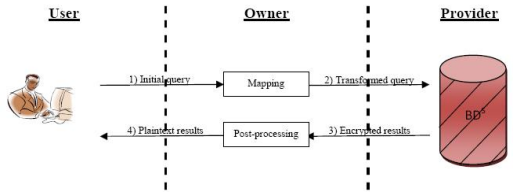

Fig. 1. Querying an outsourced database in the owner side policy enforcement

The remainder of this paper is organized as follows. Section 2 discusses the related work. We focus on securing untrusted storage and key management for outsourced databases. In section 3, we describe a new schema of database outsourcing. Section 4 reminds the joint encryption scheme which constitutes the core of our key management proposal. Then section 5 gives a preliminary description of our proposal followed in section 6 by the cloud-based key management protocol description illustrated with an example. Finally in Section 7, we discuss our proposal and give some future work.

II. Related work

Key management solutions for outsourced databases can be classified in three categories: owner side policy enforcement solutions, user-side policy enforcement solutions, and solutions where access policy is shared among actors (the owner, the user and the server provider or any other participant). In the first category [1], [12], [11], due to the owner role in the query translation and the query result filtering, the control policy is enforced by the owner itself.

978-0-7695-4105-1/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSACW.2010.47

This causes a bottleneck situation, thus reducing the benefits of outsourcing. The second category of approaches [7], [15], [5], [3], [8] relocates the access control to the user side alleviating the owner's charge in the querying process. In these solutions, the database is encrypted with symmetric keys that are delivered to the user who can manage the whole querying process, encrypting the query so that it can be evaluated over the encrypted database; then the result is retrieved by decrypting the returned response. Such solutions require to identify in a preliminary and off-line stage, for each user -or user profile-, the set of data he has access to, according to the effective access control policy. The idea is to establish a logical hierarchy among data. Several techniques are used to organize data in hierarchies. In [5], the authors consider -in order to simplify- a matrix representation of users rights on sets of data, representing the users as rows and the access configuration associated with data tuples as columns. A key derivation tree (UTH) is built from the access matrix *A* enforcing the access control policy. From an empty root vertex, the proposed algorithm starts including in the UTH all the vertices representing access configuration by scanning the matrix *A*. Moreover, the vertices should respect a partial order where each vertex *v* is pointed by an edge from a parent vertex *p* whose access configuration AC_p is included in AC_v. That is why, each vertex will be associated with a secret key *k* such that a key for vertex *v* will be given to a user *u* if and only if *u* belongs to a *v*s access configuration and *u* does not belong to *parent(v)* access configuration. Moreover based on the algorithm described in [5], in [3], the authors propose a heuristic algorithm that minimizes the total number of distributed keys delivered to all the users. In [8], the authors propose an alternative solution that, based on the same kind of rights matrix than [5], builds a binary tree [10] whose intrinsic properties contribute to reduce key management complexity. In this solution, each level of the binary tree corresponds to the rights of a user profile. Then a user is given the keys corresponding to his level in the hierarchy and can derive all the keys for allowed data using derivation mechanisms. This solution has the drawback of assigning keys in a unequal manner; few keys are assigned for user at the top of the hierarchy, while those situated in the bottom receive a bigger number of keys. To solve this problem the authors propose a preliminary sort of user profiling, depending on the right importance. This leads to minimize the average number of delivered keys. The authors of [14] propose a solution that organizes data in a binary tree and use also derivation mechanisms. The solution does not follow any defined data placement strategy to group data according to user rights, as in [5] and [8]. Consequently, as allowed data could be disseminated on different parts of the binary tree, to avoid data disclosure, the number of received keys could grow very quickly especially in a scenario of billion of data blocks as assumed in [14]. In [7], Samarati and al. initiate the shared key management solutions. In their work the authors propose a two level encryption scheme, one before outsourcing data, done by the owner and the second, in case of user or role revocation, done by the service provider.

Despite of all the efforts supplied by the previous work, the key management process remains insufficiently flexible and leads to an unavoidable database re-keying when users rights change frequently. The solution exposed in [5] supports dynamic right management but the UTH looses its intrinsic building guidelines after few changes leading to a necessary UTH rebuild and then to a whole database re-keying and key distribution. The solution in [8] supports better data right changes except in the case of data right loss. Solution in [14] suffers from a lack of flexibility in the case of right dynamics, as data hierarchy does not correspond to a logical placement. Actually, the real problem in delivering keys to the final users occurs when access rights on data are reconsidered for a user or a role. Indeed, the user, unless the database is re-keyed, continues to have access to the data. [7]'s proposal avoids data re-keying but its implementation remains complicated. On the other hand, the complexity to build an efficient hierarchy of keys is explained by the difficulty to maintain rights coherency among different user profiles due to their dependency. The contribution of this paper is twofold: First, we present a novel approach for sharing the secret between the user and a set of servers on the cloud. The main idea is : as the user does not possess the entire secret key, she can not still access data when her rights are denied; Second, as the complexity to deliver keys in a sustainable manner is explained by role dependancy, we propose to dissociate roles that are incompatible and to duplicate the target database so that only compatible roles have access to the same database copy. These core concepts are explained in next section.

III. Externalizing the database on the cloud

Existing approaches to DaaS with untrusted servers face a twofold challenge:

1) managing dynamic key distribution and
2) handling revocations.

Our first proposal to reduce key management complexity is a simple one: replicating *n* times the source database, where *n* is the number of different roles having access to the database. Each database replica is a view, entirely encrypted using the key *k* created for the corresponding role. Each time a role is created, the corresponding view is generated and encrypted with a new key expressly generated for the newly created role. This way, no changes are required on the database in case of dynamic role update. Role revocation, i.e. changing a role's access rights, leads -only in the worst case- to the creation from scratch of a new materialized view, still involving the rekeying of the corresponding database; we however argue role revocation to be a relatively unfrequent event w.r.t. user revocation. Of course, creating a replica of the database for each role introduces high redundancy, and for this reason has not been considered by previous proposals in this area. However, the trend in the storage industry has always been that as the disk capacity continues to increase, the price per storage continues to decrease. Today 1Tbyte disk drives are available

for less than 500 U.S.D. On the cloud, low cost storage does not necessarily mean low performance or reliability. Commercial database suppliers have already taken advantage of these low cost storage systems, especially by providing new storage management features. So, the assumption that role management will be handled by re-encryption is no longer a far-fetched one. Note that storage redundancy in this context could be enhanced by a deep study of role view assignment, but this point is one of our future actions. In order to avoid data re-encryption in the much more frequent case of user revocation, in our solution the key used to encrypt the role-associated database is not delivered to the users associated to the role; rather each user receives a *token* that allows her to address a cipher demand to a set KS of key servers on the cloud.

IV. Joint Encryption Schemes

In this section we shortly review some basic notions on joint encryption. The idea is that a subset of a group of parties is enabled to encrypt a given message using their own individual keys, and the cooperation of all the current participants is requested to decrypt the same message. A *joint encryption scheme* for a group of n parties, is a collection of encryption functions $\{E_S : S \subseteq [1...n]\}$ and a collection of decryption functions $\{D_i : i \in [1...n]\}$ such that given a message M, $\forall i : D_i(E_S(M)) = E_{S-\{i\}}(M)$, and M can be straightforwardly computed from $E_\emptyset(M)$. Such schemes can be *asymmetric*, when the scheme relies on an asymmetric encryption algorithm, where the E_S are easy to compute, while the D_i are hard. Here, we need the encryption algorithm $E_S()$ to be also *homomorphic*: i.e given $E_S(x)$ and $E_S(y)$, one can obtain $E_S(x+y)$ without decrypting $x+y$ for some operation $+$. Franklin and Haber [9] have shown that homomorphic joint encryption schemes can be straightforwardly derived from standard homomorphic public-key encryption schemes. Many works have considered *threshold* encryption, i.e. asymmetric joint encryption schemes where any sufficiently large subset of parties can decrypt a message. We do not deal with the threshold property in this paper, since it is not necessary for our cloud-based encryption/decryption system.

V. Preliminary description of the solution

Our approach is based on a homomorphic variation of joint encryption technique recalled in the previous section (see also [4]). The data owner generates a *virtual role key* α as a set of shadows $S_1, ..., S_n$, to be distributed across servers. Note that group schemes can be devised where these shadows are received by servers without communicating with the data owner and, potentially, without even knowing her identity. For the sake of conciseness, we shall not deal with this aspect in this paper; rather, we will use these shadows to carry out a joint encryption-decryption scheme, so that encrypted data can only be decrypted by scanning all n servers and vice-versa. In other words, the data owner generates the collection of shadows $S_1, ..., S_n$, uses them jointly to encrypt the data view, and then stores each S_i on a cloud server $KS_i \in KS$. Also, the data owner shares with the servers in KS its policy, by sending to each server KS_i the association between α, represented by the shadow S_i each server holds, and a set of users, e.g. in the form of identification tokens. Suppose now that a user *u* creates a query q (typically an attribute-value pair) and broadcasts to KS a message $E_{\beta_u}[q]$ where β is her local key, together with her token. Each server in KS independently checks if the query is compatible with the policy, i.e. verifies that the token is correctly associated to the local shadow S_i. If the token is valid, the servers in KS form a ring and jointly encrypt the message with their shadows, obtaining $E_\alpha[E_{\beta_u}[q]]$. The last server sends the encrypted query back to user *u* , who decrypts it with respect to its local key obtaining $E_\alpha[q]$. Then, user *u* sends the query to the DBMS.
The DBMS computes the result $E_\alpha[r_q]$ -where r_q is q's result- on the encrypted database and sends it to user *u*, who computes $E_{\beta_u}[E_\alpha[r_q]]$. Finally, user *u* sends this message to the server pool KS that jointly decrypts it using the shadows and bounces back $E_{\beta_u}[r_q]$, which represents the query result r_q encrypted with the user *u* local key. As the following example (Section VI-B) illustrates, the user does not have to perform anything but her usual local encryption. Whether the user will get or not access to the role data view (encrypted using the virtual role key α) depends on the policy, i.e. on the association between the user and the virtual role key, guaranteed by the user's token. In this scenario, revocation is very simple and can be performed by the data owner anytime, simply by broadcasting a signed $kill - token$ message to the servers in KS (Again, note that token killing can be performed in such a way that servers in KS ignore even the identity of the data owner)

VI. A cloud-based management protocol

In this section we first explain the basic mechanisms that our protocol relies on. We assume that a materialized view of the database is computed for each role. Our rationale is to avoid any actor (be it a user, the untrusted DBMS or one of the key servers on the cloud) to detain the whole information required to access a data view. Rather, the secret is jointly detained by the user and by the key servers on the cloud (KS). Note that servers in KS need not be trusted in any special way: in order to access the plaintext of a query q or of its result r_q, the entire KS set would need to collude [13].

A. Protocol outline

In our protocol, the database owner creates V_r, the materialized view assigned to the role r. Then, she generates a virtual key α (as a set of shadows $S_1, ..., S_n$) and uses α to encrypt V_r before outsourcing each shadow S_i to a server KS_i belonging to the set KS on the cloud. Each user u_i assigned to role *r* receives from the data owner a different key β_{u_i}, while the shadows $(S_1, ..., S_n)$ of the role key α are placed on a set KS of key servers chosen on the cloud. Whenever a user u_i associated to the role *r* wants to submit a query q, she encrypts q with the key β_{u_i}, and sends it to the first server of KS, the one detaining the S_1 shadow. The query is

NumEmp	Salary
1	10000
2	35000
3	25000
4	20000
5	15000

Fig. 2. Original table T

NumEmp	Salary
1	10000
2	35000
4	20000
5	15000

Fig. 3. The V_{r_1} materialized view

NumEmp	Salary
1	10000
3	25000
4	20000

Fig. 4. The V_{r_2} materialized view

then super-encrypted in a collaborative manner by the servers holding the set of shadows $S_1, ..., S_n$. The last server, KS_n, sends back the encrypted query to the client, who decrypts it using β_{u_i} and sends the result ($E_\alpha(q)$) to the untrusted DBMS server. The untrusted DBMS executes the query on encrypted data and sends the answer to the user u_i , who encrypts it with β_{u_i}. Then, the query is sent to the servers of KS in order to decrypt it w.r.t. α, making it readable for the final user.
Revocation of a user u_i can be performed by the data owner anytime by broadcasting a signed $kill - token(u_i)$ message to all servers in KS. Then, servers in KS will "forget" about decrypting queries encrypted with β_{u_i}. Ignoring the data owner instructions would require agreement on the part of all servers on KS. Also, in principle token killing can be performed in such a way that servers in KS ignore even the identity of the data owner.

B. Protocol execution: an example

To illustrate our proposal, let T (fig 2) be a sample table composed of five tuples t_1 to t_5. We assume that this table is assigned to two roles r_1 and r_2 as follows : r_1 gives access to tuples t_1, t_2, t_4 and t_5 while r_2 gives access to tuples t_1, t_3, t_4. The owner will then generate two materialized views V_1 and V_2, as depicted in Figure 3 and 4. The data owner generates two role keys α_1 and α_2 and encrypts V_1 {resp. V_2} with α_{V_1} {resp. α_{V_2}} before outsourcing on α_1 and α_2 on two (possibly disjoint) sets KS of cloud servers. Let us now reason with role r_1 having access to V_1. Assume also that three users u_1, u_2 and u_3 have the credentials to play role r_1. They receive respectively their personalized keys β_{u_1}, β_{u_2} and β_{u_3}.

1) Applying the protocol: In this case each user has a personalized key β_{u_i}. Let us assume u_1 sends a query q on V_1. The protocol flow goes as follows:

1) $u_1 \rightarrow KS_1 : E_{\beta_{u_1}}(q)$
2) KS_1 checktoken(u_1, r_1)
3) for $i = 1..n$, KS_i: encrypt($E_{\beta_{u_1}}(q)$, S_i)
4) $KS_n \rightarrow u_1$: $E_{\alpha_{V_1}}(E_{\beta_{u_1}}(q))$
5) u_1: decrypt($E_{\alpha_{V_1}}(E_{\beta_{u_1}}(q))$ with her personalized key $E_{\beta_{u_1}}$
6) $u_1 \rightarrow$ DBMS : $E_{\alpha_{V_1}}(q)$
7) DBMS $\rightarrow u_1 : E_{\alpha_{V_1}}(r_q)$
8) $u_1 \rightarrow KS_1 : E_{\beta_{u_1}}(E_{\alpha_{V_1}}(r_q))]$
9) for $i = 1..n$, KS_i:
   decrypt($E_{\beta_{u_1}}(E_{\alpha_{V_1} - \sum_{j=1}^{i-1} S_j}(r_q), S_i)$
   where the term $\alpha_{V_1} - \sum_{j=1}^{i-1} S_j(r_q)$ represents the partial decrypted r_q with the first $i-1$ shadows
10) $KS_n \rightarrow u_1 : E_{\beta_{u_1}}(r_q)$

Thus, u_1 will be able to decrypt the query answer r_q using its local key β_{u_1}. Suppose now that user u_1 leaves role r_1 and is associated by the data owner to another role, say r_2. All the data owner has to do is to create a new key $\beta_{u_1}^{r_2}$ and sends it to u_1. Then, the owner sends a `kill-token` notification of the old u_1 token certifying that u_1 belongs to role r_1. Upon receiving $E_{\beta_{u_1}(q)}$ if user u_1, the first server in KS detects immediately that the association no longer holds and the protocol is aborted. The revocation of u_1 does not affect users u_2 and u_3 and does not require database re-keying.

VII. Conclusion and perspectives

We have presented a technique for handling access to outsourced encrypted databases. The advantage of our technique is twofold: (1) it uses static key distribution and (2) it simplifies revocation by decoupling local keys (held by users) and protection keys (corresponding to roles).
The cloud is a suitable platform for the proposed secret sharing techniques as it provides more flexibility in choosing the key servers set. As a consequence, it is more difficult to target a special server by the attackers. Moreover, taking into account the offered servers in the cloud, it is possible to dissociate the servers set dedicated to encryption from the one dedicated to decryption which minimizes adaptative-plaintext attack risks and increases parallelism between encryption and decryption steps. Also, by using redundancy in the cloud servers, this technique could easily be extended for achieving robust encryption, i.e., recovery even in the face of network errors or lost data. The same encryption/decryption techniques could be implemented using more traditional distributed platforms but with less flexibility as there is no possible granularity control over the implied servers. We are perfectly aware that much work remains to be done before techniques like the ones proposed in this paper can be used, and that accurate implementation and performance analysis will be needed. We are currently working to validate our approach w.r.t. these issues. Also, we are very well aware that our "one-dataview-per-role" solution can lead to excessive redundancy even in presence of low (and declining) storage costs. However, hybrid techniques can be envisioned where stable core roles get their own materialized data-views, while other roles are handled in a different way, by grouping together compatible roles. Likewise, role right revocation could be enhanced to avoid data re-keying. We will address these topics in our future research.

REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[2] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–43, 2009.

[3] C. Blundo, S. Cimato, S. D. C. di Vimercati, A. D. Santis, S. Foresti, S. Paraboschi, and P. Samarati. Efficient key management for enforcing access control in outsourced scenarios. In *SEC*, pages 364–375, 2009.

[4] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In B. Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001.

[5] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective data encryption in outsourced dynamic environments. *Electronic Notes in Theoretical Computer Science*, 168:127–142, 2007.

[6] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. *Proceedings of the 10th ACM conference on Computer and communications security*, pages 93–102, 2003.

[7] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: management of access control evolution on outsourced data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 123–134. VLDB Endowment, 2007.

[8] V. El-khoury, N. Bennani, and A. M. Ouksel. Distributed key management in dynamic outsourced databases: A trie-based approach. In *DBKDA '09: Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 56–61, Washington, DC, USA, 2009. IEEE Computer Society.

[9] M. K. Franklin and S. Haber. Joint encryption and message-efficient secure computation. *J. Cryptology*, 9(4):217–232, 1996.

[10] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[11] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 29–38, 2002.

[12] H. Hacigumus and S. Mehrotra. Performance-conscious key management in encrypted databases. *Research Directions In Data And Applications Security XVIII: IFIP TC 11/WG 11.3 Eighteenth Annual Conference On Data And Applications Security, July 25-28, 2004, Sitges, Catalonia, Spain*, 2004.

[13] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[14] W. Wang, Z. Li, R. Owens, and B. Bhargava. Secure and efficient access to outsourced data. In *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 55–66, New York, NY, USA, 2009. ACM.

[15] A. Zych, M. Petković, and W. Jonker. Efficient key management for cryptographically enforced access control. *Comput. Stand. Interfaces*, 30(6):410–417, 2008.