Efficient Enforcement of Dynamic Cryptographic Access Control Policies for Outsourced Data

Anne V.D.M. Kayem
Department of Computer Science
University of Cape Town
Private Bag X3
Rondebosch, Cape Town, 7701
Email: akayem@cs.uct.ac.za

Patrick Martin and Selim G. Akl
School of Computing
Queen's university
Kingston, Ontario, Canada
K7L 3N6
Email: {martin, akl}@cs.queensu.ca

***Abstract*—Outsourcing of their data to third-party service providers is a cost-effective data management strategy for many organizations. Outsourcing, however, introduces new challenges with respect to ensuring the security and the privacy of the data. In addition to the need for standard access control policies, organizations must now be concerned with the privacy of their data and so hiding the data from the service provider is important. Simply encrypting the data before it is transmitted to the service provider is inefficient and vulnerable to security attacks when the access control policies change.**

Approaches based on two layers of encryption alleviate the privacy concern but still require re-encryption of the data when policies change. This paper presents a novel and efficient solution that employs two layers of encryption of the data and an encrypted data object containing the second access key. Changes to the access control policies are handled by re-encrypting the object containing the affected key, which is an efficient operation. The paper presents our key management approach, a security analysis of our approach, and an evaluation of the performance of a proof of concept implementation of our approach.

Keywords: Outsourced Data Management, Information Security and Privacy, Access Control, Information Retrieval

I. Introduction

Organizations outsource their data to third-party service providers to reduce their growing data management costs. The service providers in turn take care of the management aspects, granting access only to users that are authorized to do so by the data owner.

However, data outsourcing introduces concerns for confidentiality [1]. This need for confidentiality applies both to unauthorized users and the service provider (SP) because the data must be kept secret from its third party host [2]. For instance, organizations that outsource data concerning their customers' personal information need a way to protect the information even from the SP in order to maintain adequate levels of trust from their customers. Hence, the data owner must protect the data that is sent to the SP but also allow authorized users to retrieve and read the information at a later time.

A data outsourcing scenario in which the data owner transfers management of a set of data objects to a third-party SP is considered. In this data outsourcing scenario, the data owner grants access rights to subsets of the data objects to users who retrieve data objects from the SP. The asumption is that any modifications to an object are made by the users and the entire object is returned to the SP where it replaces the current copy. The SP enforces security policies to ensure that the data objects are visible only to the appropriate groups of users. Membership in these groups is dynamic so users can join and leave the groups. There are two problems that emerge here, first, the data owner wishes to keep the data secret from the SP and second, the SP needs to protect his/her integrity by guaranteeing that only authorized users get to access the data that he/she receives from the data owners.

Cryptographic access control (CAC) has been proposed as a way to solve these two security problems in data outsourcing. When cryptographic keys are used to secure the data, the data owner encrypts the data before it is transmitted to the SP. This key is then made available to all the users requiring access to the data but not to the SP. On the SP's side, to facilitate data management, the data is categorized hierarchically by imposing a second layer of encryption on the data that is guided by a hierarchical cryptographic key management (CKM) scheme. The second key is transmitted to the users requiring access to the data, according to the rules of access defined by the data owner. Therefore each user holds two keys, one that is used for authentication and the first step of decryption on the SP's end, as well as one that is used to decrypt the data into a readable format on the user's end.

The problem with this approach to securing outsourced data is that replacing keys to prevent security violations when group membership changes requires that the data owner and/or the SP re-encrypt the data with the new key to enforce data security. This procedure is expensive, particularly when large volumes of data are involved and/or the set of users with access to the data is dynamic. Therefore, a CAC scheme that circumvents the cost of re-encrypting the data when key updates occur is a desirable solution to the problem of securing outsourced data.

This paper presents a novel and efficient approach to securing outsourced data that uses three keys. For convenience, the keys are denoted B_x, K_x, and A_x where $0 \leq x \leq n-1$ and n is the maximum number of security classes in the access control hierarchy. The first key, B_x is used by the data owner

978-1-4577--1483-2/11/$26.00

to encrypt the data before it is transmitted to the SP. This satisfies the requirement that the data remains secret from all unauthorized users, including the SP. In order to categorize the data received, the SP creates a second key, K_x that the SP uses to encrypt the data received from the data owners. The third key, A_x is generated by the SP and is used to encrypt the keys, K_x. The key A_x is then transmitted to the data owner and/or the users authorized, by the data owner, to access the data. Therefore each data owner and/or authorized user holds two keys, A_x and B_x. One that is used to encrypt the data before it is transmitted to the SP, B_x; and one that is used to retrieve the data from the SP's end, A_x.

Key updates are handled by updating the key A_x and re-encrypting only the data object containing the key K_x, instead of updating K_x and re-encrypting the data that was encrypted with K_x on the SP's end. Moreover, avoiding data re-encryptions circumvents the problem of data unavailability that arises when a data owner attempts to access data while the SP is re-encrypting it with an updated key K_x.

The rest of the paper is structured as follows. In Section 2, background material on the "Database-as-a-service" paradigm is presented with a focus on the aspect of CAC to outsourced data. Section 3 presents the proposed CKM approach and Section 4, presents a security analysis of the proposed CKM approach. Section 5 presents some performance results and concluding remarks are offered in Section 6.

II. Background

The "database-as-a-service" paradigm emerged with the purpose of defining methods of outsourcing data resources to SPs on the Internet. Most solutions focus on methods of executing queries efficiently and securely on encrypted outsourced data [3]–[6]. Typically, these methods are centered around indexing information stored with outsourced data. The indexes are useful for returning responses to queries without the need to decrypt the data (or response to the query). Only the party requiring and authorized to view the response should, in principle, be able to decrypt the result returned in response their query. The challenge in developing indexing techniques is in establishing a reasonable trade-off between query efficiency, exposure to inference, and linking attacks that depend on the attacker's knowledge [7]. Additionally, there is the issue of a malicious user with access to the SP's end being able to piece together information from parts of information gathered historically and then using this knowledge to transmit false information [2], [7], [8].

Other proposals avoid this by using cryptography to protect the data both from malicious users and from the SPs. Research on using cryptography to protect access to outsourced data began with the approach that Miklau and Suciu [8] proposed in 2003. In the Miklau et al. approach, different cryptographic keys are used to encrypt different portions of an Extensible Markup Language (XML) tree by introducing special metadata nodes in the document's structure. Hence, the data remains secret to all the participants in the system and only those in possession of valid keys are able to decrypt the data. Although this approach secures the outsourced data and provides hierarchical access control, it does not address the problem of handling key updates and the need to categorize data at the SP's end.

De Capitani Di Vimercati, Foresti, Jajodia, Paraboschi, and Samariti [2], [7] build on this approach and consider the problem of authorization policy updates. The De Capitani Di Vimercati approach operates by using two keys. The first key is generated by the data owner and used to protect the data initially by encrypting the data before it is transmitted to the SP. Depending on the authorization policies the SP creates a second key that is used to selectively encrypt portions of the data to reflect policy modifications. The combination of the two layers provides an efficient and robust solution to the problem of providing data security in outsourced data environments. However, policy modifications or updates are handled by updating the affected cryptographic key and re-encrypting the data which is expensive encryption wise when large amounts of data are re-encrypted.

The literature on CAC approaches, to addressing the problem of secure data access and cost effective key management, has been investigated in the context of distributed environments like the Internet [9]–[13]. Examples of applications in which access can be controlled using these CKM approaches include, PAY-TV, sensor networks and social networking environments [14]–[16]. However, the case of controlling access to outsourced data differs from these cases in the sense that the SP needs to categorize all the data he/she receives in way that prevents malicious access and also minimizes the cost of authorization policy changes. Moreover, in the case of cryptographically supported access, when data re-encryptions involve large amounts of data, there is the risk of jeopardizing data consistency if data updates fail to be written onto a data version due to time-consuming re-encryptions.

III. Key Management

Our proposed CAC scheme is based on the concept of hierarchical CKM and so we briefly explain how hierarchical CKM schemes work before delving into our scheme. The proposed CAC scheme operates in two phases namely, the setup phase in which the keys are generated in ways that enforce the security policies and the update phase which determines how key updates are handled.

A. The Hierarchical Key Management Model

The concept of hierarchical key management builds on the lattice model of hierarchical access control that inspired multilevel access control mechanisms like the Bell-Lapadula approach [17], [18]. In the hierarchical key management model (HKM), a unique cryptographic key is assigned to each one of the classes in the hierarchy. A multilevel access control system, is enforced by allowing users, with keys belonging to higher level classes, to access information at lower levels depending on the rules of information flow in the HKM. Standard methods of granting higher level users access to

lower classes include the independent and dependent key management models.

In independent key management schemes, each security class is assigned a unique key and access to lower classes is only possible if a user at a higher class holds the required lower class key [13], [15]. The dependent key management schemes on the other hand use a series of interrelated keys and access to a lower class is only possible if a user belongs to the lower class or holds a key that allows him/her to mathematically derive the required lower class key [10], [12], [18]. The key derivation process is performed through the application of a one-way function, so that a low level key can be derived (when this is authorized) from a high level key, but the reverse is not possible.

Our proposed scheme assumes that, depending on the performance or security management benefits sought, either one of these key management models can be used. The advantage of using the independent key management approach is that it is less encryption intensive than the dependent key management approach, since data is only re-encrypted at the security class affected by the key update. However, the independent key management approach requires re-distributing the updated key to all the classes requiring the new key. On the other hand, the dependent key management approach avoids distributing many keys by changing each of the keys in the sub-hierarchy associated with the affected key. Therefore, in the worst case the whole hierarchy will need to be updated and the data re-encrypted which, as mentioned before, is costly. The choice as to which approach to adopt depends on the context with which a security administrator is faced. The other assumption is that since the keys are typically generated by a central authority, or security administrator, secure key distribution can be handled by a key exchange protocol like the Diffe-Hellman key exchange scheme [19].

B. Phase 1: Setting up the System

The data owner begins by categorizing users into exactly one of several groups (security classes) that are partially ordered. Additionally, each group U_i is associated with a data object d_i and a key B_i [18], [9], [10]. To encrypt the data before it is sent to the SP, the data owner uses the encryption function

$$E_{B_x}(d_x) = c_x$$

where $0 \leq x \leq n-1$ and n is the maximum number of classes in the hierarchy. For instance in Figure 1, a two class hierarchy with data objects d_i, and d_j that are encrypted with the keys B_i, and B_j to obtain encrypted data objects c_i, and c_j is considered.

Moreover, the data owner can facilitate key management by using a dependent key management scheme to generate the keys B_i. In this case, therefore, each group is assigned a single key that can be used to either directly decrypt and access the data, or to derive the required key. For instance, in the totally ordered hierarchy depicted in Figure 1, B_i can be used to derive the key B_j that is associated with the group

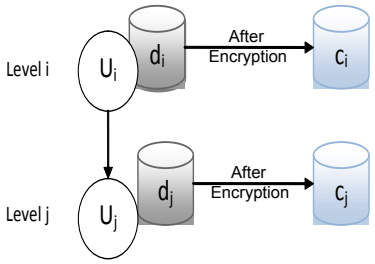

Fig. 1. An example of a totally ordered hierarchy with four security classes

below U_j. The reverse is not possible because keys associated to lower level groups cannot be used to derive keys belonging to higher level groups.

Once the data has been encrypted, the data owner transmits it to the SP for management. The SP defines a partial order hierarchy to categorize all the data that it receives. The partial order hierarchy is enforced by re-encrypting each data object received with a second key, K_i that enables the SP to authenticate and grant access only to authorized users. The encryption function that the SP uses is as follows:

$$E_{K_x}(c_x) = \beta_x$$

where $0 \leq x \leq n-1$ and n is the maximum number of classes in the hierarchy. For instance in Figure 2, a two class hierarchy with encrypted data objects c_i, and c_j that are encrypted with the keys K_i and K_j to obtain double encrypted data objects β_i, and β_j is considered. The keys K_i can also be defined

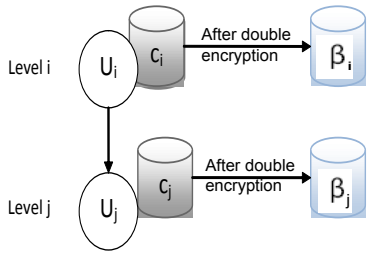

Fig. 2. Data Hierarchy on the Service Provider's End

using a dependent key management scheme, in which case each class gets assigned only a single key. The reason for this is mainly to facilitate key management. Finally, to grant users access to the data on the SP's end, the SP creates a second set of keys A_i that are used to encrypt the keys K_i using the encryption function

$$E_{A_x}(K_x) = \gamma_x$$

where $0 \leq x \leq n-1$ and n is the maximum number of classes in the hierarchy. Thus an encrypted object γ_x is obtained and can only be decrypted with the correct A_x. The SP then transmits the keys A_x to the data owner who takes care of sharing the keys with the users according to the portions of data that users are authorized to access.

For instance, as shown in Figure 3, the SP uses the keys A_i and A_j to encrypt the keys K_i, and K_j to obtain γ_i, and

γ_j. The keys A_i and A_j are then shared with the data owner who in turn will take care of distributing the keys to the users belonging to the groups U_i, and U_j respectively. So, in this case, users in group U_i get assigned the key A_i and users in group U_j get assigned the key A_j.

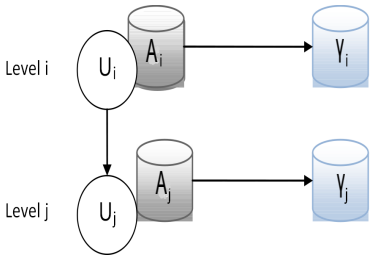

Fig. 3. Key Protection Procedure

Each user holds two keys, say A_i, B_i and the user accesses data, say d_i, by submitting their key A_i to the SP for authentication. The SP will use the key A_i to decrypt γ_i and obtain K_i that will then be used to decrypt β_i to obtain c_i that is then handed to the user. To read c_i, the user will use his/her key B_i and decrypt c_i to obtain a readable form of d_i. So in essence, the only keys a user in U_i is ever aware of are A_i and B_i.

C. Phase 2: Handling Key Updates

For simplicity, assume that key updates are triggered by changes in user group membership and the objective of key updates is to prevent a user who has left a group from subsequently accessing a data object, say d_i. In this case, the data owner will send a message to the SP to alert him/her of the change. In previous approaches, key updates imply re-encrypting all the data objects associated with the affected key. Since the cost of data encryption is directly proportional to the size of the data object, re-encrypting large files is costly and time-consuming.

Our approach overcomes this drawback by allowing the SP to replace the key A_i of the affected group and re-encrypt the key K_i. The new A_i is then sent to the users remaining in group U_i. Therefore, only the data object containing the key is re-encrypted and since this is a considerably smaller data object in comparison to the actual data, the overall cost of key updates is reduced. Since access to c_i requires that a user presents a correct A_i to the SP, once A_i is updated, the old B_i would be useless to the user who has left the system, because he/she will be unable to retrieve c_i.

IV. Security Analysis

To evaluate the theoretical security of our key management approach, three aspects are considered namely, the security of the keys that the users hold, namely A_x and B_x, the security of the data d_x, and the security of the key K_x. Our objective here is to show that no single user in the system has enough knowledge of the assigned keys to be able to deduce the true contents of the data unless they hold the "correct" keys. In fact, once the data owner encrypts and transfers the data to the SP for management, he/she can not retrieve the data without the "correct" keys, A_x and B_x.

A. Security of B_x and d_x

As mentioned in Section III-B, the key B_x is used to encrypt the data d_x before it is transmitted to the SP and transmitted to all the users that have been granted access to the data d_x. Since the key B_x is kept secret from the SP, the SP cannot read d_x, and therefore the data stays secure. This satisfies the first concern for security on the user's end, namely the wish to keep the data secret from the SP. In selecting a key generation scheme, security against collusion attack is provided by selecting a scheme that is provably secure against collusion [9], [20], [21].

Another aspect worth considering is the case in which a malicious user, say Alice steals B_x during its transmission to the users in U_x. Since Alice does not hold A_x, she will be unable to exploit her knowledge of B_x to retrieve c_x unless she presents the "correct" A_x to the SP.

B. Security of K_x

The key K_x is created by the SP and is never shared. All encryptions and decryptions with the key K_x are performed on the SP's end. Therefore, K_x remains protected.

C. Security of A_x

With respect to the security of A_x two cases are considered, in the first case a user who left a group attempts to retrieve and decrypt c_x and in the second case, a malicious user intercepts A_x and attempts to retrieve and decrypt c_x.

In the first case, to retrieve the current version of the data d_x, the user will begin by submitting his/her key, A_x, to the SP for authentication. The SP uses the supplied A_x to decrypt γ_x which is not possible because K_x has been re-encrypted with an updated A_x. Therefore, even though the key B_x was not updated, the user will be unable to retrieve c_x and decrypt it to read d_x.

The second case involves a malicious user attempting to gain access to d_x by intercepting A_x. As shown in Figure 4, the malicious user, say Alice, intercepts A_x during its transmission to the users in group U_x. Alice then presents A_x to the SP and since A_x is correct, the SP retrieves K_x from the decryption of γ_x. The K_x obtained is then used to decrypt β_x and retrieve c_x that is returned to Alice. However, since c_x needs to be further decrypted using B_x which Alice does not hold, obtaining c_x is useless.

Therefore, our approach to key management, gives a strong guarantee of securing outsourced data. The only case in which Alice can hope to retrieve d_x, is that in which she steals both A_x and B_x. However, if A_x is updated fairly frequently the cost of stealing the keys should outweigh its benefits. Moreover, a secure key exchange protocol based on an asymmetric cryptography technique [22], could also be used to circumvent key interception attacks.

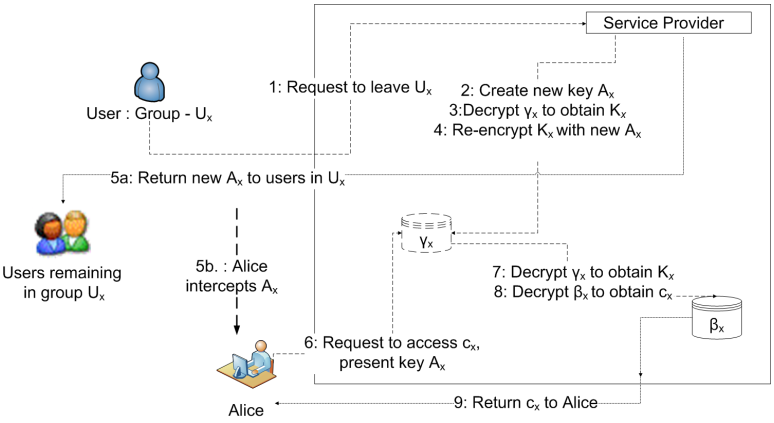

Fig. 4. Malicious Attempt to Retrieve d_x

V. Performance Analysis

The scalability of the proposed CAC scheme to controlling access to outsourced data is evaluated with a set of experiments conducted on an IBM Pentium IV computer with an Intel 3.00Ghz processor and 1GB of RAM.

A. Comparison Strategies

The performance of our proposed CAC scheme is compared to a naive approach and the approach proposed by De Capitani Di Vimercati et al.

1) Naive Approach: Our naive approach gives basically a worst case situation in which the data owner encrypts the data and sends it to the SP. The data is not re-encrypted on the SP's end. The key management model is based on a series of partially ordered interdependent keys and so every time a user group's membership changes the data owner reacts by updating the keys throughout the entire hierarchy, re-encrypting the data associated with the affected keys and re-transmitting the data to the SP. This approach is costly encryption wise and insecure because the key updates, data re-transmissions and key distributions create more opportunities for security attacks. This case is a baseline and the CAC scheme used is the one that Akl and Taylor proposed [18]. The reason for this choice is that the Akl and Taylor CAC is an example of a scheme that requires updating keys throughout the entire hierarchy in reaction to key updates.

2) De Capitani Di Vimercati et al: In this case the approach proposed by De Capitani Di Vimercati et al. [1] is implemented using the scheme proposed by Atallah, Frikken and Blanton [9] to handle key management. Here the data is encrypted on the data owner's end and transmitted to the SP where a second layer of encryption is applied to the data. Key updates are handled by updating only the key associated with the data as well as the keys that are used to authenticate higher level users before granting them access to the data.

3) Proposed Key Management Approach: Our proposed CAC scheme is implemented as outlined in Section III-B, and use the Akl and Taylor scheme [18] to handle key management. This is to show that our design approach to handling key management in outsourced data scenarios is such that key updates are handled effectively even in the worst case scenario.

B. Comparison Metrics

For clarity, some definitions of the performance evaluation terminology used is presented after which a description of the experimental platform is given.

- ***Cost of System Setup:*** This is the cost of key generation and data encryption both on the SP's end and the data owner's end. This cost is the total time (in minutes per hierarchy size) that it takes to set up the system with respect to the size of the file.
- ***Cost of Key Updates:*** This is the cost (time it takes in minutes per file size) of re-establishing security when a user group's membership changes and the system needs to update the keys to prevent the departed user from continuing to access the system.

C. Experimental Setup

All three key management approaches were implemented on a Microsoft Windows XP platform using the Java 2 Standard Development Kit and Eclipse [23]–[26]. While our proof of concept implementation uses Triple Data Encryption Standard (Triple DES) keys for the encryption and decryption procedures, it can easily be replaced by the current Advanced Encryption Standard (AES) standard [27]. In Triple DES the key size is 168 bits and the block size 64 bits. However, due to the meet-in-the-middle attack the effective security Tripe DES provides is only 112 bits [28], [29]. In the experiments we use hierarchies with 3-73 security classes (groups). The hierarchies are formed in a way that allows roughly 2-10 sub-groups to be directly connected to a group. The hierarchy sizes and shapes are based on the computational limitations of the machine that we used for the experiments. It is also worth noting that different structures of hierarchies might yield slightly different results depending on the time it takes to select a valid set of exponents to use in generating the keys.

The cost of system setup is evaluated for the different hierarchy sizes with a file of size $\approx$ 32MB. We also evaluate

the encryption overhead generated in setting up the system with respect to the size of a file. In this case we used files of sizes 1KB,...,100MB. The experiment on cost of key updates is first performed with a file of size $\approx$ 32MB and then with varied files sizes of 1KB,...,100MB. In the first case using a standard file size of $\approx$ 32MB allows us compare the cost of updates with respect to different file sizes. While the second case allows us focus on the encryption overhead generated, in handling key updates to reflect authorization policy changes, with respect to varied file sizes. Henceforth, we use "the Vimercati approach" to refer to the method that De Capitani Di Vimercati et al. proposed [1].

D. Results

Our performance analysis is centered on two aspects: *cost of system setup* and *cost of key updates* to reflect authorization policy changes. In the following sections we present results to highlight the performance enhancements offered by our proposed approach in comparison to a naive or baseline case and the Vimercati approach.

1) Cost of System Setup: The experiments here evaluate the time it takes each of the three key management approaches that we described in Section V-A to generate the required keys and encrypt the data associated with every security class in the hierarchy, both on the data owner's and the SP's ends. The standard deviation for each point plotted is ± 0.83 minutes.

As shown in Figure 5 the cost of generating keys in our approach during system setup is highest in comparison to the naive and Vimercati approaches since our scheme generates three keys instead of two as the other schemes do. We note, however, that the cost of key generation in our proposed scheme is reasonable since it takes less than a minute to generate keys for a hierarchy of 73 security classes. In fact, a closer look at Figure 5 and Figure 6 indicate, that data encryption is the more cost intensive operation and that the the costs of key generation and data encryption grow linearly with the hierarchy size.

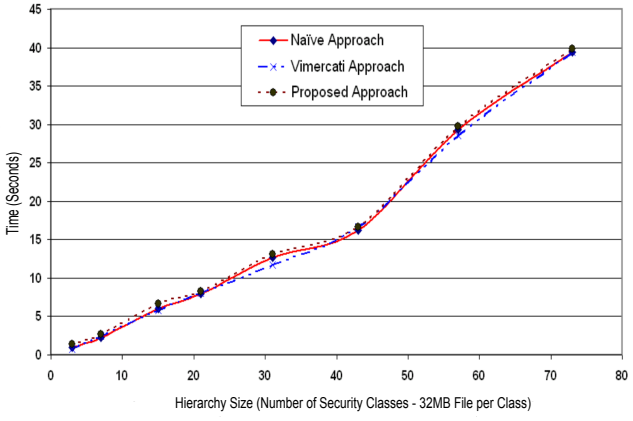

Fig. 5. System Setup: Cost of Key Generation Only per Hierarchy

For smaller sized hierarchies it can be noted that the setup costs are relatively similar while larger hierarchies (i.e. 73 security classes) create a wider divergence between our proposed scheme and the other two. This is because the cost of generating the third key and encrypting the associated key files has a bigger impact on the overall cost as the size of the hierarchy grows.

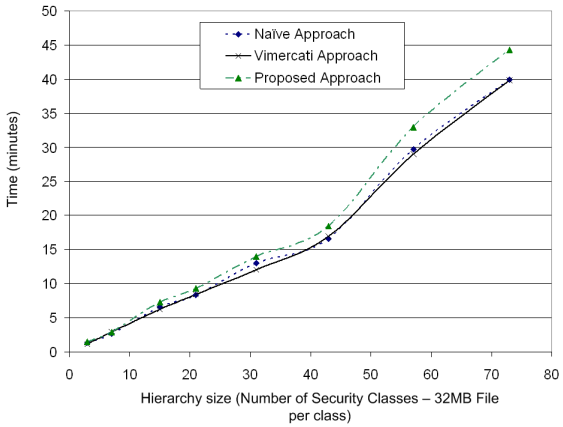

Fig. 6. System Setup: Cost of Key Generation and Data Encryption per Hierarchy

Our second experiment evaluates the encryption overhead generated in setting up one security class on the system. As shown in Figure 7, in this case we consider the effect of different file sizes on the cost of data encryption using each of the three approaches.

As expected, we note that with all the three approaches the overhead generated during encryption is proportional to the size of the file. The encryption overhead generated using our proposed scheme is marginally higher than that in the Vimercati and naive approaches. However a closer look indicates that the difference is roughly 0.1 minutes which is due to having to encrypt the key file on top of encrypting the data files.

2) Cost of Key Updates: The experiments in this section evaluate the time it takes our proposed key management approach to generate a new key in response to a key update request. In the first experiment, we consider the worst case

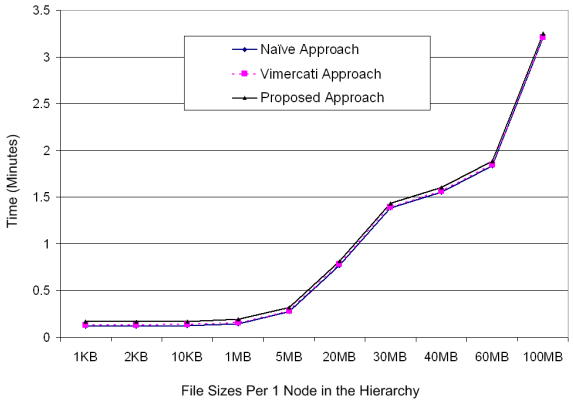

Fig. 7. Encryption Overhead: Cost of system setup with respect to file size

in which keys throughout the whole hierarchy need to be

updated. The performance of our approach was compared to the Vimercati approach and the naive approach. The standard deviation for each point plotted is ± 0.83 minutes in the Vimercati and naive approaches and ± 0.5 minutes in our proposed approach.

As shown in Figure 8 the cost of key updates in the naive approach remains unchanged from the cost of setting up the hierarchy and is the most costly of the three approaches. It is also worth noting at this point that this approach is the most insecure because the data is exposed briefly each time a key is updated. Additionally, since the data needs to be constantly re-transmitted to the SP, frequent key updates open up more possibilities for malicious interceptions.

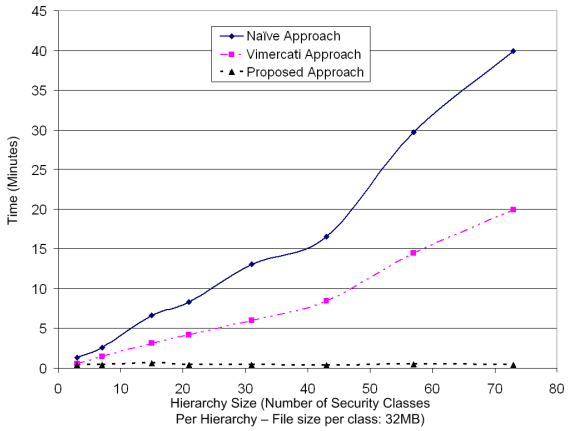

Fig. 8. Key Updates: Cost of Replacements and Re-encryptions

The Vimercati approach performs much better than the naive approach, which is to be expected since the encryption time is cut roughly by a half. Only the keys on the SP's end need to be updated and this cost grows linearly with the hierarchy size.

Our proposed approach outperforms the previous two because updates are handled by replacing the key A_x and re-encrypting the key K_x, associated with each class in the hierarchy. This key file is smaller than the data files, hence the gains in performance.

Our second key update experiment looks at the impact of file sizes on the overhead generated during key updates. In this case a single security class in the hierarchy that is associated with a file of size 1KB,...,100MB, is considered. As shown in Figure 9, the observation is that the performance of the three approaches is similar for the smaller files, i.e. 1KB,...,1MB.

This confirms our observation that the size of the file is directly proportional to the encryption overhead generated. However, as the files get larger the performance of the naive and Vimercati approaches grow worse while our proposed approach's performance stays fairly constant.

While the cost of updates is proportional to the file and hierarchy size, the differences in cost are more significant in the Vimercati and naive approaches than in the proposed approach. This is because the key file size is fairly constant and so the encryption costs relatively the same.

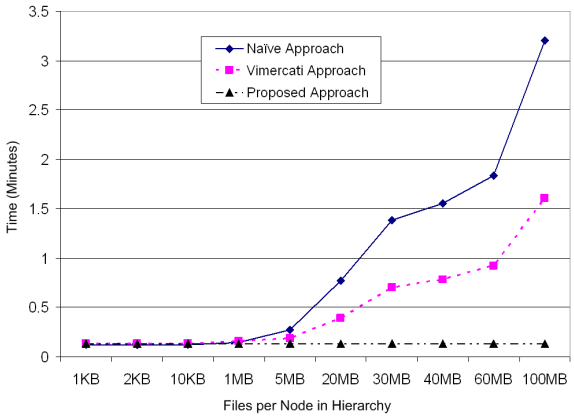

Fig. 9. Key Updates: Cost of Replacements and Re-encryptions per file size

VI. Conclusions

In this paper, a solution is presented to the problem of handling authorization policy modifications when access control is enforced cryptographically in situations where data is stored and offered to clients by an external server. The discussion of the background on this topic showed that most solutions have tended to focus on indexing techniques that strike a balance between query efficiency and controlling exposure to inference attacks. However, a key concern in outsourced data environments is that of protecting the secrecy of the data effectively. Data owners want guarantees that their data will not be read and/or modified by the SPs and SPs need some way of categorizing the data to facilitate data management while ensuring that malicious access is prevented.

More recently, De Capitani Di Vimercati et al. [1], [2] proposed a solution to addressing the concern for data security on both the data owner's and SP's ends. Their approach uses a two layered encryption model that allows the data owner to encrypt the data before it is transmitted to the SP where a second encryption layer is applied to the data. However, their solution faces the drawback that key updates are handled by requiring the SP to generate a new key and re-encrypt the affected data. Moreover, as shown in Figure 9 since re-encrypting large data objects is time consuming, issues related to copy consistency are likely to arise if users attempt to write updates on to the data during the encryption.

This problem was addressed by designing an access control approach that is supported by three cryptographic keys, A_x, B_x, and K_x. The keys, B_x and K_x are used to protect the data from both unauthorized users and the SP by allowing the data owner and SP to apply two layers of encryption. The last key, A_x, is used to encrypt the key, K_x, that the SP used to apply the second layer of encryption to the data and is distributed to the users authorized to access the data. To access the outsourced data, a user needs to present the key, A_x to the SP for authentication. Once this test is passed the user is handed a data object that can only be decrypted into a readable format if the user holds the key with which it was encrypted. So in essence, each user holds two keys,

one that authenticates them to the SP, A_x, and another key, B_x, that is used to read the data that they retrieve. When a situation occurs that requires a change in the authorization policy, the SP reacts by creating a new key, A_x, to replace the old key, re-encrypts the key file containing K_x, and transmits the key, A_x to the users authorized to access the affected data. Since the key file is small in comparison to the data file, the cost of key updates is reduced considerably in comparison to previous approaches. An added advantage of our approach is that it removes the requirement of having to re-encrypt the data each time the authorization policy changes and therefore circumvents the problem of data consistency.

The results obtained from our performance analysis indicate that while the cost of setting up the system is marginally higher than that in the De Capitani Di Vimercati et al. approach, the gains afforded by our key update approach overcome this drawback. Our security analysis also shows that our approach is secure against attacks that might be carried out by malicious users trying to guess at or deduce the contents of the data.

REFERENCES

[1] S. De Capitani Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samariti. Over-encryption: Management of access control evolution on outsourced data. In *In Proc. VLDB 2007*, pages 123–134. Vienna, Austria, September 23-28 2007.

[2] S. De Capitani Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samariti. A data outsourcing architecture combining cryptography and access control. In *Proc. of the 2007 ACM Workshop on Computer Communications Security (CSAW)*, pages 63–69. Fairfax, Virginia, USA, November 2 2007.

[3] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of ACM SIGMOD*, pages 563–574. Paris, France, June 2004.

[4] G. Aggrawal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Proc. CIDR 2005*, pages 186–199. Asillomar, California, USA, 2005.

[5] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of the 18th ICDE Conf.*, pages 29–38. San Jose, California, USA, 2002.

[6] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Encrypting sql over encrypted data in the database-service-provider model. In *Proc. of ACM SIGMOD*, pages 216–227. Madison, Wisconsin, USA, June 2002.

[7] A. Ceselli, E. Damiani, and S. De Capitani Di Vimercati. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. on Information and System Security*, 8(1):119–152, February 2005.

[8] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. of the 29th VLDB Conf.*, pages 898–909. Berlin, Germany, September 2003.

[9] M. J. Atallah, K. B. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *Proc. ACM Conference on Computer and Communications Security*, pages 190–202. Alexandria, Virginia, USA, November 7-11 2005.

[10] M. J. Atallah, M. Blanton, and K.B. Frikken. Key management for non-tree access hierarchies. In *Proc. of ACM Symposium on Access Control Models and Technologies*, pages 11–18. Lake Tahoe, California, USA, June 7-9 2006.

[11] Jason Crampton. Cryptographically-enforced hierarchical access control with multiple keys. In *Proc. of the 12th Nordic Workshop on Secure IT Systems (NordSec 2007)*, pages 49–60, 2007.

[12] A.V.D.M. Kayem, S.G. Akl, and P. Martin. On replacing cryptographic keys in hierarchical key management systems. *Journal of Computer Security*, 16(3):289–309, 2008.

[13] R.H. Hassen, A. Bouabaallah, H. Bettahar, and Y. Challal. Key management for content access control in a hierarchy. *Computer Networks*, 1(51):3197–3219, 2007.

[14] J.C. Birget, X. Zou, G. Noubir, and B. Ramamurthy. Hierarchy-based access control in distributed environments. In *Proc. of the IEEE International Conference on Communications, Vol. 1*, pages 229–233. Helsinki, Finland, June 11-14 2001.

[15] W. Yu, Y. Sun, and R. Liu. Optimizing the rekeying cost for contributory group key agreement schemes. *IEEE Transactions on Dependable and Secure Computing*, 4(3):228–242, July-Sept.

[16] S. Zhu, S. Setia, and S. Jajodia. Performance optimizations for group key management schemes. In *Proc. of 23^{rd} International Conference on Distributed Computing Systems (ICDCS '03*, pages 163–171. Fairfax, Virginia, USA, May 19-22 2003.

[17] D. E. Bell and L. J. Lapadula. Secure computer systems: Mathematical foundations. *MITRE Report, MTR2547*, March 1973.

[18] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, August 1983.

[19] Wikipedia Diffe-Hellman. Diffe-hellman key exchange. http://en.wikipedia.org/wiki/Diffie-Hellman, March 2009.

[20] A.V.D.M. Kayem, S.G. Akl, and P. Martin. Heuristics for improving cryptographic key assignment in a hierarchy. In *Proc. of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, pages 531–536. Niagara Falls, Canada, May 21-23 2007.

[21] S. J. Mackinnon, P. D.Taylor, H. Meijer, and S. G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, c-34(9):797–802, September 1985.

[22] Martin E. Hellman. An overview of public key cryptography. *IEEE Communications Magazine, 50th Anniversary Commemoration Issue*, 40(5):42–49, May 2002.

[23] Java. Java development kit (software). http://en.wikipedia.org/wiki/Java-Development-Kit, 2010.

[24] Java. Java (software). http://www.java.com/en/download/index.jsp, 2011.

[25] Eclipse. Eclipse (software). http://en.wikipedia.org/wiki/Eclipse-(software), 2010.

[26] Eclipse. Eclipse (software). http://www.eclipse.org/downloads/packages/, 2010.

[27] J. Daemen and V. Rijmen. The block cipher rijndael. *J.-J. Quisquater and B. Schneier (Eds.): CARDIS 2000, LNCS 1820*, pages 277–284, 2000.

[28] Wikipedia Triple DES. Triple des. http://en.wikipedia.org/wiki/Triple-DES, March 2011.

[29] Gaël Rouvroy, Francois-Xavier Standaert, Jean-Jacques Quisquater, and Jean-Didier Legat. Design strategies and modified descriptions to optimize cipher fpga implementations: fast and compact results for des and triple-des. In *Proceedings of the 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, FPGA '03, pages 247–247, New York, NY, USA, 2003. ACM.