An Improved Scheme for Protecting Medical Data in Public Clouds

Nikos Fotiou and George Xylomenos

Mobile Multimedia Laboratory
Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Patision 76, Athens 10434, Greece
{fotiou,xgeorge}@aueb.gr

Abstract. Public Clouds offer a convenient way for storing and sharing the large amounts of medical data that are generated by, for example, wearable health monitoring devices. Nevertheless, using a public infrastructure raises significant security and privacy concerns. Even if the data are stored in an encrypted form, the data owner should share some information with the Cloud provider in order to enable the latter to perform access control; given the high sensitivity of medical data, even such limited information may jeopardize end-user privacy. In this paper we employ an access control delegation scheme to enable the users themselves to perform access control on their data, even though these are stored in a public Cloud. In our scheme access control policies are evaluated by a user-controlled gateway and Cloud providers are only entrusted with respecting the gateway's decision. Furthermore, since medical data must often be shared with health providers of the user's choice, we rely on a proxy re-encryption technique to allow such sharing to take place. Our scheme encrypts data before storing them in the Cloud and applies proxy re-encryption using Cloud resources to encrypt data separately for each (authorized) user. Our proxy re-encryption scheme ensures that misbehaving Cloud providers cannot use re-encryption keys to share content with unauthorized clients, while delegating the costly re-encryption operations to the Cloud.

Keywords: access control, proxy re-encryption, medical data, public clouds

1 INTRODUCTION

Nowadays, smart devices that collect users' vital signals have become a commodity. It is expected that the data collected by these devices will soon be used for preventing and/or diagnosing various health related problems, as well as for promoting a healthier way of living and well-being. Storing and sharing these data using a public Cloud infrastructure appears to be an appealing option, as public Clouds offer cost effective, reliable and always-on storage services. On the

other hand, security and privacy concerns are raised, as medical data are highly sensitive and they should be very well protected, even against misbehavior by the Cloud service provider. Encryption and access control can be used as a countermeasure, but privacy threats remain. For example, an access control policy of the form "these (encrypted) data can only be accessed by psychiatrist A" reveals to the entity that performs access control that the data owner shares some data with a psychiatrist.

In this paper we propose a scheme that allows secure and private storage of medical records in the Cloud. Our scheme allows data owners to define access control policies and to enforce them by themselves. The Cloud provider is only responsible for storing data and for respecting the access control decisions of the data owner. Even if the Cloud provider misbehaves, the data remain protected, since they are encrypted so that only authorized users can access them; unauthorized users – including the Cloud provider – can learn nothing about the data. In order to achieve our goal we use the system proposed by [7] by adding an additional layer of data confidentiality protection.

Since our proposal encrypts data before storing them in the Cloud, they cannot be directly shared with authorized health providers, without revealing the user's encryption keys. To allow controlled data sharing, our scheme relies on re-encrypting the data before sharing, so that they can only be decrypted by users authorized by the data owner. Rather than having the user's devices re-encrypt data, we rely on a proxy re-encryption scheme so as to delegate this processing to the Cloud provider, without however allowing the Cloud provider to gain access to the encrypted data. In this manner, the user only needs to deal with the original data encryption, delegating all further storage and processing to the Cloud provider.

This paper extends our previously published work [8] in the following areas: (i) we improve our proxy re-encryption based scheme so as to protect our system against misbehaving Cloud providers, (ii) we add a client authentication procedure, (iii) we provide more details about our protocol, (iv) we perform a more thorough evaluation of our system, including its security evaluation.

The remainder of the paper is organized as follows. Section 2 briefly presents access control delegation and proxy re-encryption. Section 3 presents our system design in detail. In Section 4 we evaluate our solution and in Section 5 we present related work in the area. Finally, we conclude our paper in Section 6

2 BACKGROUND

2.1 Access control delegation

The access control scheme proposed in [7] separates *data storage* and *access control* functions: the former is implemented in a public Cloud, whereas the latter is implemented by a trusted entity named the *access control provider* (ACP). These entities interact with each other as follows (Figure 1)[1]: Initially, a data

[1] The description has been modified to fit the purposes of the present paper.

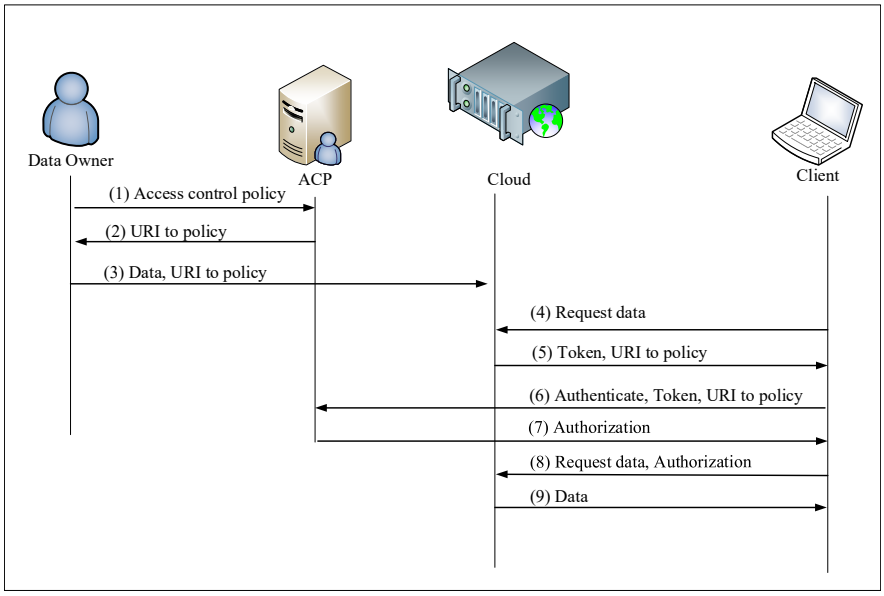

Fig. 1. Access control delegation (reproduced from [8])

owner creates an *access control policy*, stores it in an ACP (step 1) and obtains a *URI* for that policy (step 2). Then, he stores some data in the Cloud, indicating at the same time the URI of the policy that protects these data (step 3). When a client tries to access these data (step 4), the Cloud responds with the URI of the access control policy and a unique token (step 5). Then, the client authenticates herself to the ACP and requests authorization (step 6). If the client "satisfies" the access control policy, the ACP generates a signed *authorization* and sends it back (step 7). Finally, the client repeats her request to the Cloud, this time including the authorization (step 8). The Cloud checks the validity of the authorization and if it is valid, it returns the desired data (step 9).

This scheme has many advantages. The Cloud provider learns nothing about the client since all her personal data (which are required to evaluate the access control policy) are stored in the ACP. Moreover, Cloud providers do not have to interpret any access control policies, therefore they do not need to understand content owner specific semantics. Each ACP can implement any conceivable access control policy, since the Cloud provider only sees the URI identifying the policy and the authorizations returned by the ACP. Access control policies are reusable i.e, in order to protect a new item using an existing access control policy the same URI can be simply re-used. Access control policies can be easily updated; updating an access control policy does not involve any communication with the Cloud provider. Finally, provided that many Cloud providers support this scheme, it is trivial for a data owner to migrate from one Cloud provider to another, as the URIs of the access control policies remain the same.

2.2 Proxy re-encryption

A Proxy re-encryption (PRE) scheme is a scheme in which a third, semi-trusted party, the *proxy*, is allowed to alter a ciphertext encrypted with the public key of a user A (the delegator), in a way that another user B (the delegatee) can decrypt it with her own appropriate key (in most cases, her secret private key). During this process the proxy learns nothing about the private keys of A and B, and does not gain access to the encrypted data.

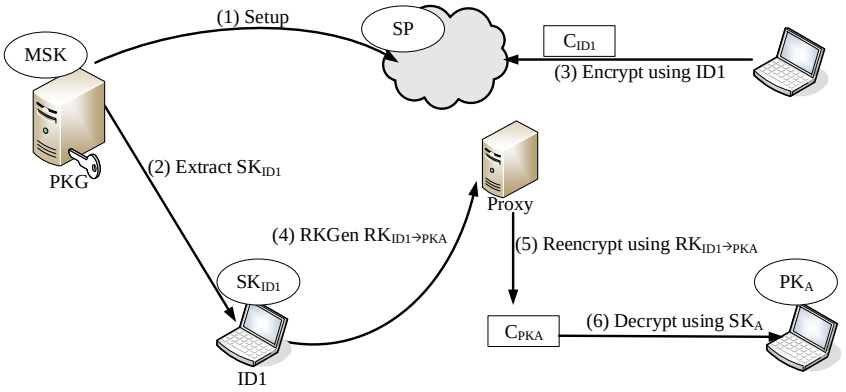

Fig. 2. IB-PRE example (adapted from [8])

In this paper we employ the *identity-based proxy re-encryption* (IB-PRE) by Green and Ateniese [11]. In particular we use a variant of that scheme in which the delegatee uses public key based encryption (PKE) instead of identity-based encryption (section 5 of [11]). This scheme specifies the following algorithms (the description has been adapted to the PKE variant):

- `Setup`: it is executed by a *Private Key Generator* (PKG). It takes as input a security parameter k and returns a `master-secret key` (MSK) and some `system parameters` (SP). The MSK is kept secret by the PKG, whereas the SP are made publicly available.
- `Extract`: it is executed by a PKG. It takes as input the SP, the MSK, and an identity ID, and returns a `secret key` SK_{ID}. An ID can be any arbitrary string.
- `Encrypt`: it can be executed by anyone. It takes as input an identity ID, a message M, and the SP, and returns a ciphertext C_{ID}. This ciphertext can only be decrypted by the owner of SK_{ID}, i.e., the secret key that corresponds to the identity ID.
- `RKGen`: it is executed by the owner of the identity $ID1$. It takes as input the SP, the secret key SK_{ID1} and the PKE public key PK_A of a user A. It outputs a re-encryption key $RK_{ID1 \to PK_A}$.

– `Reencrypt`: it is executed by a *proxy*. It takes as input the SP, a re-encryption key $RK_{ID1 \rightarrow PK_A}$, and a ciphertext C_{ID1} and outputs a new ciphertext C_{PK}, which can be decrypted by the owner of the PKE secret key SK_A.
– `Decrypt`: it is executed by the owner of the PKE secret key SK_A. It takes as input SP, C_{PK}, and SK_A, and returns the message M.

Figure 2 gives an example of a complete IB-PRE transaction. In this figure, initially the PKG generates the MSK and the SP, and makes the SP publicly available, while keeping the MSK to itself (step 1). This initializes the system. When a user $ID1$ wants to use the system, it asks the PKG to extract the secret key SK_{ID1} and return it to user $ID1$ (step 2). Another user $ID3$ can then encrypt a piece of text using the publicly known identity of $ID1$, creating a ciphertext C_{ID1}, which is then stored in a proxy (step 3). This ciphertext can only be decrypted by the user that owns $ID1$, and therefore knows the corresponding SK_{ID1}. To allow another user $ID2$ to decrypt the content using a PKE private key SK_{RSA_2}, the owner of $ID1$ creates a re-encryption key $RK_{ID1 \rightarrow RSA2}$ using the well known PKE public key PK_{RSA_2} of user $ID2$ and sends it to the proxy (step 4). The proxy re-encrypts C_{ID1} using the re-encryption key and generates C_{ID2} (step 5). The owner of SK_{RSA_2} is now able to decrypt the re-encrypted ciphertext (step 6). The proxy learns nothing about the contents of the ciphertext or the secret keys of the users.

If the original version of the scheme is used (instead of the PKE variant described above) then the secret keys of delegatees should be generated by a PKG. This however raises security concerns, since the PKG will know the private keys of all users. Although this is not a problem in some scenarios, in our case a delegatee is a doctor or a hospital that has access to very sensitive information. Therefore, this is an unacceptable security threat. Moreover, if a delegatee interacts with many delegators (as, for example, in the case of a hospital that interacts with its patients) then this results in a non-negligible key management overhead. For this reason, we rely instead on the PKE keys of the delegatees for the re-encryption procedures.

3 DESIGN

In this section, we explain how the aforementioned access control and proxy re-encryption schemes are adapted for our scheme. We assume the use of smart devices that collect user related data, such as smart watches that measure cardio activity. All collected data are stored in a public Cloud. The smart devices do not interact directly with the Cloud; they instead communicate with a user controlled *gateway*. This gateway holds the roles of both the PKG and the ACP described in the previous section, i.e., the gateway generates the appropriate secret keys and is responsible for enforcing access control policies. In addition, a gateway is responsible for initially encrypting (not re-encrypting) files, storing them in the Cloud, and for generating re-encryption keys. Clients interested in receiving a file stored in the Cloud, initially send an unauthorized request to the Cloud provider. The Cloud provider re-directs them to the appropriate gateway,

where they authenticate themselves and get authorized to access the protected file. The authorization process also results in the creation of an appropriate re-encryption key, which is securely transmitted to the Cloud provider. Then, clients issue authorized requests and receive the (re-encrypted) file. All communications (between the smart devices and the gateway, between the gateway and the Cloud, between the clients and the Cloud, and between the clients and the gateway) are secured using TLS. Figure 3 gives an overview of our system entities and their interactions, which we explain in more detail in the following subsections.

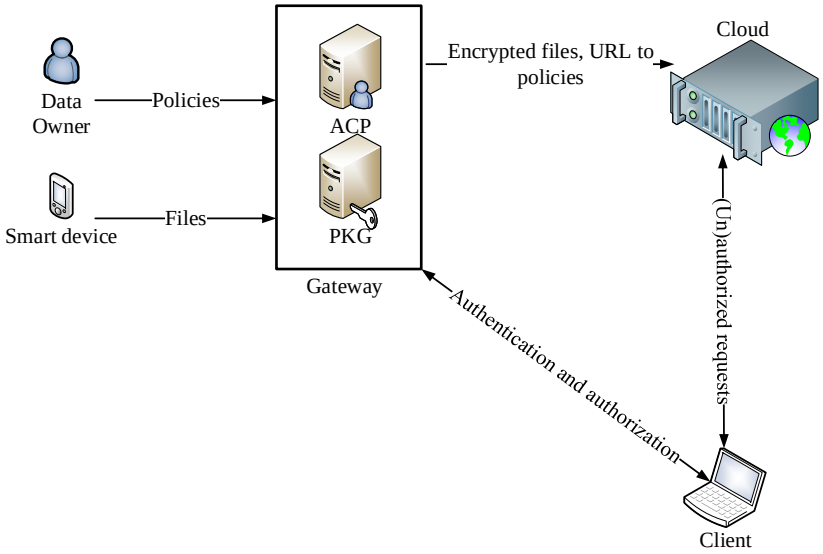

Fig. 3. An overview of the entities of our scheme and their interactions (adapted from [8])

3.1 Setup

With this procedure a data owner creates access control policies and generates the appropriate cryptographic keys. Our system uses NIST's core Role-based Access Control (core-RBAC) model [6]. Based on this model, data owners create tables of clients, roles, and permissions. The table of clients contains tuples of the form $[index, identity]$ where $index$ is an integer number unique for each client and $identity$ is the client's public key. The table of roles contains tuples of the form $[role, < client_{index} >]$, where $role$ is a unique role name, and $< client_{index} >$ is a list of indices from the clients table and represents the clients that hold that specific role. Finally, for each file, the client maintains a table of permissions that contain tuples of the form $[operation, < role >]$, where

operation is an operation over the file (e.g., read, write, delete) and $< role >$ is a list of roles that are permitted to perform that operation. Each permission table is identified by a unique URI_{policy}. All relationships in a core-RBAC model are many-to-many, hence, a client may have multiple roles and a role may be allowed to perform multiple operations.

Table of clients

Index	Identity
001	PK_{001}
002	PK_{002}
003	PK_{003}

Table of roles

Role	Clients
My Doctors	001
My Friends	002,003
Emergency	001,003

Permission Table
https://gw.example.com/policy32B

Operation	Roles
Read	My Doctors, Emergency
Modify	My Doctors

Fig. 4. Access control policy example

Figure 4 illustrates an example of access control policy definition. In this example, a data owner has defined three clients and three roles. It can be observed that client 003 has multiple roles. The data owner has created a permissions table and has specified its URI_{policy}. We can observe in this table that the role "My Doctors" is allowed to perform multiple operations.

The first time the setup procedure is executed, the gateway executes the IB-PRE `setup` algorithms and generates a master secret key (MSK) and the corresponding (public) system parameters. The MSK is securely stored in the gateway.

3.2 Data storage

Data storage in the Cloud is achieved using the following steps:

- For each file that arrives in the gateway a permissions table and an appropriate URI_{policy} is generated, or an existing one may be re-used.
- The gateway generates a symmetric encryption key K, encrypts the file using K (we refer to the output of this encryption as $Enc_K(file)$), and encrypts K

using the IB-PRE *encrypt* algorithm, using URI_{policy} as the input identity (we refer to the output of this encryption as $C_{URI_{policy}}(K)$).
– The gateway stores $Enc_K(file)$, $C_{URI_{policy}}(K)$, and URI_{policy} in the Cloud.

Gateways keep track of all files and their associated URI_{policy} and Cloud provider in a *table of files*. A table of files contains a set of tuples of the form $[filename, policy, Pub_{CP}]$ where $filename$ is the file name, *policy* is the URI_{policy} of the file's permissions table, and Pub_{CP} is the public key of the Cloud provider where the file is stored. Cloud providers maintain a similar table that contains entries of the form $[filename, policy, Pub_{GW}]$, where Pub_{GW} is the public key of the gateway.

3.3 Unauthorized request

This procedure is executed by a client in order to perform an operation over some protected data. The client sends an operation request to the Cloud provider that contains nothing but the operation itself and the name of the file it concerns. Upon receiving the request the Cloud provider creates a unique *token* (i.e., an adequately large random number) and sends it back to the client, along with the corresponding URI_{policy}. Cloud providers keep track of all generated tokens, as well as their associated URI_{policy}.

3.4 Client authentication and authorization

This procedure is executed by a client upon receiving a response to an unauthorized request. The client generates an authorization request that is composed of the following fields: URI_{policy}, *token*, $filename$ (i.e., the name of the desired file), *operation*, Pub_{CP}, Pub_{Client} (i.e., the public key of the client), and digitally signs this message using his private key (we refer to the outcome of the signature operation as $Sign_{Client}(msg)$). Then, the client sends the authorization request to the gateway denoted by URI_{policy}. Upon receiving this request, the gateway performs the following steps in order to authorize the client:

– It verifies $Sign_{Client}(msg)$. If the signature verification fails, the client is not authorized, as he has not proved his identity.
– From the files table it retrieves the *policy* and the Pub_{CP} of the entry that corresponds to the $filename$ included in the authorization request and checks if these fields match those included in the request. If this verification fails, the client is not authorized.
– It retrieves the file's permissions table and examines if the client identified by Pub_{Client} has a role authorized to perform the operation included in the authorization request. If this verification fails, the client is not authorized.

If the client is authorized, then the gateway performs the following operations:

– It executes the IB-PRE `RKGen` algorithm and creates $RK_{URI_{policy} \rightarrow Pub_{client}}$ and encrypts this key using Pub_{CP}. We refer to the output of the latter encryption as $C_{CP}(RK)$.

– It sends to the client an authorization response which contains $C_{CP}(RK)$ and a digital signature generated using the gateway's private key that covers $C_{CP}(RK)$ and all fields of the authorization request, except Pub_{Client} and $Sign_{Client}(msg)$. We refer to that signature as $Sign_{GW}(msg)$.

3.5 Authorized request

This procedure is executed by an authorized client. The client constructs an authorized request that includes the following fields: the $filename$ of the desired file, the *operation*, the *token* received with the execution of the *unauthorized request* procedure, and the authorization response retrieved with the execution of the *client authentication and authorization* procedure. Then, the client sends this request to the Cloud provider. Upon receiving this request, the Cloud provider performs the following steps in order to decide if the client is permitted to perform the requested operation:

– It retrieves URI_{policy} and Pub_{GW} that corresponds to the *name* and examines if the retrieved URI_{policy} matches the one associated with the token. If it does not match, the client is authorized.
– It evaluates $Sign_{GW}(msg)$. If the signature verification fails, the client is not authorized.

If the client is permitted to perform the requested operation, the Cloud provider performs the following operations:

– It uses $RK_{URI_{policy} \rightarrow Pub_{client}}$ and the IB-PRE `Reencrypt` algorithm to re-encrypt $C_{URI_{policy}}(K)$ so as to generate $C_{Client}(K)$.
– It sends $C_{Client}(K)$ and $Enc_K(file)$ back to the client.

Figure 5 illustrates a message sequence diagram of the unauthorized request, client authentication and authorization, and authorized request procedures.

3.6 External roles

When protecting medical data stored in the Cloud, it is desirable to have roles, and create access control policies based on such roles, which are defined by external (third) parties. For example, "doctors of hospital A" could be such a role, defined by the entity "hospital A". Contemporary cryptographic techniques such as attributed-based encryption [10], or hierarchical identity-based encryption [3] could be used to achieve this goal. However, we do not consider this option, because, for security reasons, we want each client to be able to generate her keys by herself, which is not possible with these cryptographic techniques. Moreover, these cryptographic techniques have been found to be ineffective when used for controlling access to data stored in the Cloud [9]. Instead, we follow a more conservative approach. We assume that each role is identified by a public key, generated by the same third party that has defined this role. This key is used by data owners in the table of roles instead of $< client_{index} >$. Moreover, the public

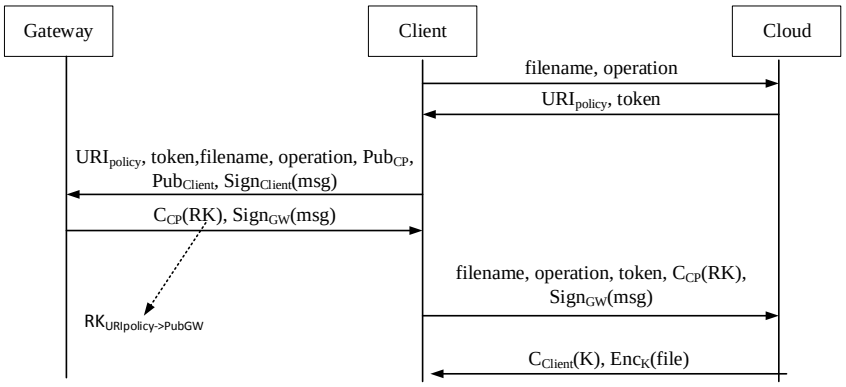

Fig. 5. Message sequence diagram

key of each client is included in a X.509 certificate which is digitally signed using the private key of the role. For instance, in our example the public keys of the doctors should be signed by the private key of the role "doctors of hospital A". If a client has multiple roles, he should have multiple X.509 certificates.

When a client requests authorization from the gateway, she includes in her request her digital certificate. The digital signature included in the certificate is used by the gateway in order to evaluate whether or not the client belongs to an authorized role. If this is the case, then the gateway can use the public key of the client (included in the certificate) to generate the appropriate re-encryption key (as described in the previous section), and therefore to allow her to access the protected file. Note that the gateway does not need to know or store any details about the members of an external role; it only needs to know the public key of that role. In our example, this allows a hospital to change the set of doctors that it has authorized, without communicating with all the gateways of the clients that trust the hospital.

4 EVALUATION

We have implemented the IB-PRE part of our system by modifying the Green-Ateniese IB-PRE implementation included in the Charm Crypto library [2] to support PKE for the delegatee. In particular, we have added support for the Cramer and Shoup elliptic curve based public key cryptosystem [4]. That is, in our implementation a ciphertext generated using Identity-based encryption is transformed into a ciphertext that can be decrypted using a Cramer-Shoup secret key (combined with some pairing operations).

In order to achieve a security level equivalent to RSA with a key size of 1024 bits for the encryption of the symmetric key, the size of SP is 2048 bits, the size of $C_{URI_{policy}}(K)$ is 2048 bits, and the size of a re-encryption key is 2816

bits. In Table 1 we report the time required to perform various cryptographic operations, in an Xbuntu 14.04 Desktop machine, running in a single core of an Intel i5-4440 3.1 GHz processor with 2GB of RAM, using the Charm Crypto library v0.43, python v2.7, the pbc library v0.5.4, and the gmp library v5.1.3.

Table 1. Computation overhead of IB-PRE cryptographic operations

Operation	**time in ms**
Create $C_{URI_{policy}}(K)$	23
Generate $RK_{URI_{policy} \rightarrow Pub_{client}}$	40
Re-encrypt $C_{URI_{policy}}(K)$	4
Decrypt $C_{Client}(K)$	25

The IB-PRE cryptographic algorithm used by our system has been proven to be secure in [11]. Each data item is encrypted using a different symmetric encryption key, therefore the compromise of a symmetric encryption key would require the re-encryption of only that specific item with a fresh key. This is an inevitable overhead of all similar systems and it is due to the fact that public key encryption cannot be applied directly to the file contents, due to its computational complexity. Nevertheless, for small data items, such as readings from wearable devices, it may be possible to negate the need for symmetric encryption.

Traditional proxy re-encryption schemes require proxies to be semi-trusted, i.e., a proxy should (i) not share re-encryption keys, and (ii) use re-encryption keys only for authorized users. Our system relaxes the second requirement: since symmetric encryption keys are encrypted using URI_{policy} as an input identity a $RK_{URI_{policy} \rightarrow Pub_{client}}$ would be useful only for clients that abide by URI_{policy}. In other words, if a client does not abide by an access control policy, the gateway will never generate the corresponding re-encryption key.

Client revocation is achieved by removing a client from a role, or by removing a role from a permission. From the moment a client is disallowed to perform an operation, a gateway will not generate a re-encryption key when that client requests authorization for this operation. Nevertheless, if the Cloud provider caches re-encryption keys and it is not trusted to use them properly, then the URI_{policy} of the permission table from which the client has been removed, has to be updated. As a consequence, a new $C_{URI_{policy}}(K)$ will have to be generated.

Our system borrows most of the properties of the access control delegation scheme described in [7]. In particular, our system is generic enough, it can be easily implemented by a Cloud provider, data can be easily transferred between Cloud providers that implement this solution, it protects client privacy against Cloud providers, and it allows easy modification of access control policies. Compared to [7], our system does not hide the Client's interests from the ACP. This happens because we use URI_{policy} for protecting content confidentiality, hence it is not possible to use a different URI_{policy} for each operation (as in [7]). If hiding the Client's interests is highly desirable, then the "level extension" (sec. 3.5 in [7]) can be used.

Another notable difference of the present system compared to [7] is that the present system does not rely on an external mechanism for authenticating clients to ACPs, using instead a digital signature (i.e., $Sign_{Client}(msg)$). In order to prevent malicious users from re-using an authentication message, ACPs should keep track of already seen tokens. Nevertheless, even if this not possible, or this check fails, the malicious user will end up receiving a file that he cannot access.

5 RELATED WORK

Löhr et al. [14] have proposed a solution for securing e-health clouds based on *Trusted Virtual Domains* (TVDs). TVD is a virtualization technique that creates secure "sandboxes" where user data can reside. This solution is orthogonal to our system: the solution by Löhr et al. concerns the design of secure clouds specific to e-health services, whereas our solution assumes a generic cloud service and builds a secure data sharing system on top of it.

Wu et al. [17] propose an access control mechanism for sharing electronic health records in the Cloud. The main component of their mechanism is an *access broker* that is responsible for enforcing access control policies. The access broker is an entity shared among many stakeholders, therefore, privacy concerns are raised. In our work, access control policies are enforced by data owners in a way that reveals no information about data owners or clients to third parties (including the Cloud provider). Son et al. [15] propose a mechanism that supports "dynamic" access control, i.e., access control that takes into consideration the user's context. In their solution, access control is also implemented in the Cloud, therefore the same privacy concerns are raised.

Fabian et al. [5] use attribute-based encryption (ABE) to protect medical data stored in multi-Cloud environments and shared among different cooperative organizations. ABE produces encrypted data in a way that only users with specific "attributes" can decrypt. In essence, ABE incorporates access control policies into ciphertexts. The disadvantage of using ABE for this purpose is that the loss of a private key that corresponds to an attribute requires the generation of a new key, the distribution of this key to all users that have this attribute, and the appropriate encryption of all files protected by this attribute. In contrast, in our system the loss of the data owner's secret key only requires a new encryption of all symmetric keys. Similarly, [1], [12], [13] use attribute-based encryption to protect personal health records stored in public cloud environments; these solution also suffer from the same problems.

Thilakanathan et al. [16] use ElGamal public key encryption and a proxy re-encryption like protocol to protect generic health data stored in the cloud. Their solution relies on a centralized trusted third party that generates private keys on behalf of users. In our system users generate their private keys by themselves, therefore our approach offers increased security.

6 CONCLUSION AND FUTURE WORK

In this paper we presented a scheme that allows secure and privacy preserving storage of medical data in public Clouds. Our solution combines access control delegation and proxy re-encryption, providing content confidentiality, client privacy enhancement, and resilience against malicious entities. This is achieved by following a gateway-based design, where a user-controlled gateway is responsible for encrypting user generated data, authenticating clients and enforcing access control policies. Cloud providers learn no information about the identity of the clients accessing the protected data and they are only trusted have to respect the gateway's decisions. Moreover, our proxy re-encryption based confidentiality solution protects sensitive data against misbehaving Cloud providers, even those that do not respect the gateway's access control decisions. Our proof of concept implementation shows that our solution is feasible, posing minimal overhead.

Future work involves the transfer of the encryption process to the devices that generate the data. In this manner, the device could store the data directly to the Cloud, avoiding the gateway, therefore reducing communication overhead. In this setup, the gateway would still hold the ACP and PKG roles. Moreover, the ACP could also be used for authenticating end-user devices to the Cloud.

References

1. Akinyele, J.A., Pagano, M.W., Green, M.D., Lehmann, C.U., Peterson, Z.N., Rubin, A.D.: Securing electronic medical records using attribute-based encryption on mobile devices. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 75–86 (2011)
2. Akinyele, J., Garman, C., Miers, I., Pagano, M., Rushanan, M., Green, M., Rubin, A.: Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering 3(2), 111–128 (2013)
3. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005, Lecture Notes in Computer Science, vol. 3494, pp. 440–456. Springer Berlin Heidelberg (2005)
4. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) Advances in Cryptology – CRYPTO '98, Lecture Notes in Computer Science, vol. 1462, pp. 13–25. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
5. Fabian, B., Ermakova, T., Junghanns, P.: Collaborative and secure sharing of healthcare data in multi-clouds. Information Systems 48, 132 – 150 (2015)
6. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (Aug 2001)
7. Fotiou, N., Machas, A., Polyzos, G.C., Xylomenos, G.: Access control as a service for the cloud. Journal of Internet Services and Applications 6(1), 1–15 (2015)
8. Fotiou, N., Xylomenos, G.: Protecting medical data stored in public clouds. In: Proceedings of the 2nd International Conference on Information and Communication Technologies for Ageing Well and e-Health (ICT4AWE) (2016)

9. Garrison III, W.C., Shull, A., Myers, S., Lee, A.J.: On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In: Proceedings of the IEEE Symposium on Security and Privacy (2016)
10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 89–98 (2006)
11. Green, M., Ateniese, G.: Identity-based proxy re-encryption. In: Katz, J., Yung, M. (eds.) Applied Cryptography and Network Security, Lecture Notes in Computer Science, vol. 4521, pp. 288–306. Springer Berlin Heidelberg (2007)
12. Li, M., Yu, S., Zheng, Y., Ren, K., Lou, W.: Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Transactions on Parallel and Distributed Systems 24(1), 131–143 (Jan 2013)
13. Liu, J., Huang, X., Liu, J.K.: Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption. Future Generation Computer Systems 52, 67 – 76 (2015)
14. Löhr, H., Sadeghi, A.R., Winandy, M.: Securing the e-health cloud. In: Proceedings of the 1st ACM International Health Informatics Symposium. pp. 220–229 (2010)
15. Son, J., Kim, J.D., Na, H.S., Baik, D.K.: Dynamic access control model for privacy preserving personalized healthcare in cloud environment. Technology and Health Care 24(1), 123–129 (2015)
16. Thilakanathan, D., Chen, S., Nepal, S., Calvo, R., Alem, L.: A platform for secure monitoring and sharing of generic health data in the cloud. Future Generation Computer Systems 35, 102 – 113 (2014)
17. Wu, R., Ahn, G.J., Hu, H.: Secure sharing of electronic health records in clouds. In: Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). pp. 711–718 (Oct 2012)