Key Management as a Service

Liran Lerman[1,2], Olivier Markowitch[1] and Jorge Nakahara Jr[1]
[1]*Quality and Security of Information Systems, Département d'informatique, Université Libre de Bruxelles, Belgium*
[2]*Machine Learning Group, Département d'informatique, Université Libre de Bruxelles, Belgium*

Keywords: Cloud Computing : Key Management : Threshold Cryptosystem : Protocol

Abstract: In this paper we consider the security issues related to the key management in cloud computing. We focus on the difficulty of managing cryptographic keys necessary to maintain for example the confidentiality of information stored in the clouds. In this framework, we present a threshold cryptosystem as well as three protocols, based on cooperation between cloud providers and a random number generator which is a trusted third party, that covers the issue of key management.

1 INTRODUCTION

The advent of cloud computing[1] can be seen as a natural step in the evolution of on-demand information technology services and products. Indeed we find in cloud computing concepts from grid computing, virtualization, etc.

International Data Conseil (IDC) estimated that cloud computing services represented 5% of global IT investment in 2009 and could catch 10% in 2013 (European Union authorities estimate a similar growth)(informatique, 2010). It seems that we are not facing a fad but rather a new stage of the Internet evolution.

There are many benefits in moving to the clouds. For example, many small and medium enterprises (SME) consider the adoption of cloud computing solutions in order to reduce their investment in hardware, software and support, as well as to optimize the management of their needed resources; moreover they have a feeling of security in case of disasters (Hogben, 2009; Sogeti, 2009).

On the other hand, the concerns of companies are mainly focused on data security issues (Mather et al., 2009), and more precisely issues related to confidentiality[2], availability, integrity, repudiation and loss of data/services (Hogben, 2009; Catteddu and Hogben, 2009; Gellman, 2009; Mather et al., 2009)[3]. These issues rely indeed on the fact that users' data and applications reside on an infrastructure owned by a third party resulting in a loss of control of their data. This shortcoming can be mitigated by using cryptographic techniques.

Following (Geater, 2011), "*more people are using more cryptographic keys than ever before, and cryptographic is meaningless without strong key management*". It is indeed obvious that without a strong key management, applied cryptography is useless. We have to create, store and remove cryptographic keys properly.

In this paper we focus on long-term cryptographic keys (e.g. keys that are used to ensure the security of data stored on cloud infrastructures, rather than on short term keys such as ones in the Transport Layer Security for example). Furthermore, for the sake of simplicity, we focus here on the key management in the context of data encryption.

There are many possible key management protocols in the framework of cloud computing.

First of all, we can transpose as is all the key management into the cloud infrastructure. In other words, all the generation, storage, and replacement of keys would be realized by the cloud provider itself. In this case, cloud providers still have access to sensitive encrypted data and keys to decrypt it. Trust is therefore the biggest issue as quoted in (Urquhart, 2009). Even if there are access control and Service Level Agreement (SLA) (Kandukuri et al., 2009) which can stipulate that no one can decrypt data without authoriza-

[1]E.g. Google (Google App Engine), Microsoft (Microsoft Azure), IBM (IBM Smart Business Service), Amazon (Amazon EC2), VMWARE (VMWARE vCloud), EMC (EMC Atmos), Salesforce, etc.

[2]Confidentiality appears in a Top 10 obstacles to and opportunities for growth of cloud computing (Armbrust et al., 2009).

[3]Security is considered as a concern for 74.6% of North American companies interested in cloud computing (Sogeti, 2009).

tion, there are many risks such as loss of isolations between users (named tenants)[4] (Ristenpart et al., 2009; Catteddu and Hogben, 2009), malicious employees within the cloud provider (Rocha and Correia, 2011; Catteddu and Hogben, 2009), inefficient or unreliable removal of data[5] (Catteddu and Hogben, 2009), information leakage from indexing (Squicciarini et al., 2010), etc.

On the other hand, a cloud consumer can store its encrypted data on a cloud infrastructure and its cryptographic keys in another cloud infrastructure (optionally in many others). In this case, an opponent has to access several cloud providers in order to obtain the decrypted information. The drawback of this solution is that it forces the consumer to manage several contracts, one for each cloud provider.

Another solution is to keep all key management in the customer's data center. In this case, keys are stored on-premise and encrypted data are stored off-premise. It is probably the safest solution if we have a good local key management, but the resulting architecture is not a complete cloud service as defined by ENISA (Catteddu and Hogben, 2009) (e.g. *shared resources*) and it definitely contradicts the key concept of cloud computing: an access to a service from everywhere.

Yet another solution is to keep a part of the key management on-premise and another part off-premise: for each encrypted data, the decryption key could be divided, for example, in two subkeys, one stored on the cloud provider and another on-premise. Another example is when each key stored in the cloud provider is encrypted by (a single key) keys stored on-premise. In this case, no one in the cloud can find the secret key without having access to the key management on-premise. However, these two solutions are probably the worst in terms of storage, customers having to store keys on-premise and off-premise.

In brief, we can see that we are facing a dilemma specific to the cloud: we do not want to store keys on-premise neither on the cloud but we want to secure our strategic data which can be on a server also accessed by competitors. As quoted in (Mather et al., 2009), "*proper key management is a complex and difficult task*" in the cloud. The aim of this paper is to propose solutions for this problem of key management that are based both on the inter-cloud approach and on threshold cryptosystem. The proposed solutions are not specific to SaaS (Software as a service), PaaS (Platform as a service) or IaaS (Infrastructure as a service). They can be implemented in each of these types of cloud service but in the case of SaaS the cloud provider has to implement them, while in the cases of PaaS and IaaS the customer has to realize these implementations.

The paper is organized as follows. In section 2, we introduce briefly our solution and the key concepts. In section 3 we detail our solution and we conclude in section 4.

2 OVERVIEW OF THE SOLUTION

In this section, we present an overview of our solution. Our aim is to increase the confidence of users towards cloud providers but in a way that ensures that neither a cloud provider nor an opponent can have access to (decrypted) users' information. For this, our solution is based on three concepts which will be detailed in the next sections: interCloud, threshold cryptosystem and Fast Random Number Generator.

2.1 INTERCLOUD

The interCloud is an interconnection of cloud providers as the internet became a network of networks. It was first presented in 2007 by Kevin Kelly (key,) and detailed in (Bernstein et al., 2009; Celesti et al., 2010; Buyya et al., 2010).

A metaphor of the interCloud is the insurance field. Each insurance can insure rare events affecting a small part of a place. However, they cannot insure alone against a risk of tsunami (for example). In this special case, the event is rare but it affects a large part of their customers. In order to confront these events, insurers join in a group allowing them to share the risks.

The situation is similar when considering cloud infrastructures. Cloud providers could be grouped (e.g. in order to share the risks of saturation of resources). We base our solution on this concept and call it Key Management as a Service (KMaaS)[6].

[4]Not all virtualization softwares are bug-free.

[5]Security problems may happen when a client wants to remove an information stored within his cloud provider. The cloud computing hardware being also accessible to other tenants, the cloud provider must be careful that the deleted information are not made available to other clients (at the time of a memory allocation for example). In addition, copies of the same data can be performed by the cloud provider for issues of availability and redundancy; and because removing data in all places where it is stored is not necessarily trivial, deleting an information often results in dereferencing the corresponding data rather than a physical removal. Logfiles could also allow to recover deleted information.

[6]We are aware that many challenges have to be sur-

2.2 THRESHOLD CRYPTOSYSTEMS

Our solution is also based on a new $(t, n+1)$-threshold cryptosystem.

Following the classical idea, we can fit a unique polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ of degree $t-1$ with t points where the first coefficient a_0 is the secret information shared between n cloud providers (symbolized by $CC_1, CC_2, ..., CC_n$) (Shamir, 1979). Note that a_0 and a_{t-1} must be nonzero. In order to share it, the random number generator (RNG) provides the secret value of $f(i)$ and the public value of i to the i^{th} cloud provider and the values $f(n+1)$ and $n+1$ to the user.

Assume that a customer U has to send a secret information $m \in \mathbb{N} \setminus \{0\}$ to his cloud provider CC_1. Let $A \xrightarrow{p} B : x$ denotes Alice sending an information x to Bob through a private channel (e.g. secured by Transport Layer Security) and $A \rightarrow B : x$ with a public channel. Let also CC_i denotes the i^{th} cloud provider. Our symmetric encryption threshold cryptosystem works as follows:

- Let $g(i) = \prod_{j \in \{1;n+1\} \setminus \{i\}} \frac{j}{j-i}$
- The RNG generates t random $\{a_0, a_1, ..., a_{t-1}\}$ and constructs the polynomial $f(x)$
- RNG $\xrightarrow{p} CC_i : \{i; f(i) = K_{CC_i}\}\ \forall i \in \{1, 2, ..., n\}$
- RNG $\xrightarrow{p} U : \{n+1; f(n+1) = K_U\}$
- $U \xrightarrow{p} CC_1 : m + f(n+1)g(n+1) = E_{K_U}(m) = m_1$
- $CC_1 \xrightarrow{p} CC_2 : m_1 + f(1)g(1) = E_{K_{CC_1}}(m_1) = m_2$
- $\ldots$
- $CC_n \xrightarrow{p} CC_1 : m_n + f(n)g(n) = E_{K_{CC_n}}(m_n) = m + a_0$

$E_K(m)$ is the One Time pad encryption of a message m using a secret key K. In order to be consistent with the threshold cryptosystem paradigm we use n cloud providers in order to encrypt a data while at least t cloud providers are sufficient.

Finally, CC_1 has $m + a_0$ and a part of the shared key $K_{CC_1} = f(1)$, while the i^{th} cloud provider has $K_{CC_i} = f(i)$. The decryption protocol applies the threshold cryptosystem backwards. On the basis of the $f(i)$ secretly kept by t (where $1 \leq t \leq n$) cloud providers the cryptosystems reconstructs the polynomial needed to obtain a_0 and subtracting it to $m + a0$. More precisely the i^{th} cloud provider computes $E_{K_{CC_i}}(m_i) = m_i + f(i)g(i)$ during the encryption process and $D_{K_{CC_i}}(m_i) = m_i - f(i)g(i)$ during the decryption process. Note that $g(i)$ depends on the number of cloud providers involved in the encryption/decryption process. Therefore, at the time of the encryption, adding n terms is equivalent to only add t terms (the additional added terms do not change the polynomial). In other words t distinct points (shares) are enough to interpolate a polynomial of degree $t-1$. The additional $n-t$ points will fall in the already known polynomial curve and so will not lead to a different polynomial.

Note that the RNG should not keep nor leak the shares or the polynomial coefficients. In other words it does not need to store anything to allow protocols to run. Moreover $f(n+1)$ and $f(0)$ must have at least the same length than m.

Each time a customer has to send a confidential information to its cloud provider, encryptions of this information have to be realized by n cloud providers. Therefore we focused on a solution based on a symmetric algorithm for performance reasons.

Furthermore, cloud computing involves a mass consumption of cryptography. In order to generate the $n+1$ keys each time when the customer has to encrypt a data, we must have a fast RNG. Indeed the new symmetric threshold cryptosystem E_k is a variant of the One Time Pad which involves one random key shared between n entities.

The main advantage of this cryptosystem is that it is a commutative symmetric $(t, n+1)$-threshold cryptosystem (i.e. $E_{CC_i}(E_{CC_j}(m)) = E_{CC_j}(E_{CC_i}(m))$) which is essential for the following protocols. Furthermore it allows to encrypt/decrypt a data as long as there are at least t available honest cloud providers. On the other hand the confidentiality of a data is assured as long as there are less than t dishonest cloud providers. Therefore the security of the scheme is based on the One Time pad which is unconditionally secure ; furthermore the private communication channels prevent any leak of keys or messages without the commitment of cloud providers.

3 PROTOCOLS

In this section we propose three protocols intended to address the security issues previously described.

In order to keep our protocols clear and readable, we assume that the entities involved in the protocols are authenticated and that the message integrity is guaranteed during the transactions.

The ideas behind our protocols are manifold:

1. we assume that each cloud provider will offer some services to the other cloud providers
2. the cloud providers are trusted to behave accordingly to the protocol (i.e. they provide correctly

passed before the advent of interCloud but research such in (Bernstein et al., 2009) show progression in this field.

their share) but they are not trusted to stay out of a coalition. Therefore we consider than no more than t cloud providers would try to collude

3. we will use the $(t, n+1)$-threshold cryptosystem, presented in section 2.2, in order to spread encrypted data among several cloud providers (say n cloud providers) in such a way that $t \leq n+1$ cloud providers are enough to decrypt the data
4. each encrypted information is signed in order to ensure that all cloud providers have participated in the encryption process and therefore to ensure it is not CC_1 who made the whole protocol alone
5. The RNG is a trusted third party. Its role is similar as to one of a dealer in secret sharing schemes.

3.1 PROTOCOL 1

In the first protocol we assume that the cloud provider CC_1 keeps most of the keys while the user U has only to store and secure one asymmetric private key. Furthermore the encrypted information as well as digital signatures related to these information are stored on CC_1

To encrypt an information m, a (t,n)-threshold cryptosystem similar to the one seen previously is used. The main difference is that the user is not involved in the threshold cryptosystem.

Let u_m be an identification token of the message m (e.g. $u_m = h(m) \parallel U \parallel CC \parallel d$, where h is a collision resistant hash function, $\parallel$ denotes concatenation of bitstrings and d represents a time stamp).

The protocol is as follows:

- RNG $\xrightarrow{p} CC_i : \{i; K_{CC_i}\}; \forall i \in [1,n]$
- $U \xrightarrow{p} CC_1 : E_{k_U}(m); u_m$
- $CC_1 \xrightarrow{p} CC_2 : E_{K_{CC_1}}(E_{k_U}(m));$
  $sig_{CC_1}(u_m); E_{k_U}(1 \parallel K_{CC_1})$
- $CC_2 \xrightarrow{p} CC_3 : E_{K_{CC_2}}(E_{K_{CC_1}}(E_{k_U}(m)));$
  $sig_{CC_2}(sig_{CC_1}(u_m)); E_{k_U}(1 \parallel K_{CC_1}); E_{k_U}(2 \parallel K_{CC_2})$
- ...
- $CC_n \xrightarrow{p} CC_1 : E_{K_{CC_n}}(\ldots E_{K_{CC_1}}(E_{k_U}(m))\ldots);$
  $sig_{CC_n}(\ldots sig_{CC_1}(u_m)\ldots);$
  $E_{k_U}(1 \parallel K_{CC_1}); \ldots; E_{k_U}(n \parallel K_{CC_n})$

In this protocol $E_{k_U}(m)$ represents an asymmetric encryption of a message m under the user's public key k_U. Note that $sig_A(u_m)$ is the digital signature realized by Alice (using her signature private key) on u_m. This allows to the user to verify that every cloud provider has participated in the encryption process. Furthermore notice that CC_1 signs u_m and not the secret message m in order to give minimum information to an attacker about m.

When the user wants to access his information, the decryption works as follows:

- $U \xrightarrow{p} CC_1 : u_m$
- $CC_1 \xrightarrow{p} U : E_{K_{CC_{|t|}}}(\ldots E_{K_{CC_{|1|}}}(E_{k_U}(m))\ldots);$
  $E_{k_U}(|t| \parallel K_{CC_{|t|}}); \ldots; E_{k_U}(|1| \parallel K_{CC_{|1|}})$

Thanks to the encryption scheme the user has to remove only t encryption layers among n. Note that $CC_{|j|}$ denotes one among the n cloud providers. In other words $CC_{|j|}$ is not specifically the jth cloud provider (i.e. $CC_{|j|} = CC_{i_j}$ where $i_j \in \{2, ..., n\}$).

The user must store only one key whereas in order to retrieve a secret message, an opponent has to attack U and CC_1. It is similar to the situation where we have one cloud provider and one user. The message m is stored in an encryption form in CC_1 helped by key k_U. Nevertheless we present this protocol as a first step in the design of our solution of key management.

3.2 PROTOCOL 2

In this second protocol the cloud provider CC_1 has to store a signature, a portion of the key and the encrypted information while the other providers, as well as the user U have to store a portion of the key.

The encryption protocol works as follows:

- RNG $\xrightarrow{p} CC_i : \{i; K_{CC_i}\}; \forall i \in [1,n]$
- RNG $\xrightarrow{p} U : \{n+1; K_U\}$
- $U \xrightarrow{p} CC_1 : E_{K_U}(m); u_m$
- $CC_1 \xrightarrow{p} CC_2 : E_{K_{CC_1}}(E_{K_U}(m)); sig_{CC_1}(u_m); u_m$
- $CC_2 \xrightarrow{p} CC_3 : E_{K_{CC_2}}(E_{K_{CC_1}}(E_{K_U}(m)));$
  $sig_{CC_2}(sig_{CC_1}(u_m)); u_m$
- ...
- $CC_n \xrightarrow{p} CC_1 :$
  $E_{K_{CC_n}}(E_{K_{CC_{n-1}}}(\ldots E_{K_{CC_1}}(E_{K_U}(m))\ldots));$
  $sig_{CC_n}(sig_{CC_{n-1}}(\ldots sig_{CC_1}(u_m)\ldots)); u_m$

When the user wants to access his information, the decryption works as follows:

- $U \rightarrow CC_1 : u_m$
- $CC_1 \xrightarrow{p} CC_{|t|} :$
  $E_{K_{CC_{|t|}}}(E_{K_{CC_{|t-1|}}}(\ldots E_{K_{CC_{|2|}}}(E_{K_U}(m))\ldots)); u_m$
- $CC_{|t|} \xrightarrow{p} CC_{|t-1|} :$
  $E_{K_{CC_{|t-1|}}}(\ldots E_{K_{CC_{|2|}}}(E_{K_U}(m))\ldots); u_m$
- ...
- $CC_{|2|} \xrightarrow{p} U : E_{K_U}(m); u_m$

The commutative encryption scheme used in this protocol allows each chosen cloud provider i among n cloud providers to remove his encryption layer.

Compared to the previous protocol, in order to recover an information m, an attacker must attack U and $CC_{|i|}$ $\forall i \in [1,t]$.

A drawback is that if more than n-t cloud providers disappear after the encryption phase, the encrypted information cannot be deciphered anymore. Therefore, the values of t and n shall be carefully chosen to strike a balance between the minimum number of t of honest shareholders (to avoid coalitions of dishonest players), and the number n of total shareholder to avoid disruption of the protocol.

The next protocol, which is the added value of our paper, shows how we can have the same security that this protocol but without having the user to keep a key locally.

3.3 PROTOCOL 3

In this last protocol, the aim is to have only the cloud providers that store and secure cryptographic keys. The cloud provider CC_1 has to store encrypted information, a portion of the key and a signature while the other providers have to store a portion of the key. The user U does not store any key.

The protocol works as follows:

- RNG $\xrightarrow{p} CC_i : \{i; K_{CC_i}\}; \forall i \in [1,n]$
- RNG $\xrightarrow{p} U : \{n+1; K_U\}$
- $U \xrightarrow{p} CC_1 : E_{K_U}(m); u_m$
- $CC_1 \xrightarrow{p} CC_2 : E_{K_{CC_1}}(E_{K_U}(m)); sig_{CC_1}(u_m); u_m$
- $CC_2 \xrightarrow{p} CC_3 : E_{K_{CC_2}}(E_{K_{CC_1}}(E_{K_U}(m)))$; $sig_{CC_2}(sig_{CC_1}(u_m)); u_m$
- ...
- $CC_n \xrightarrow{p} CC_1 :$ $E_{K_{CC_n}}(E_{K_{CC_{n-1}}}(\ldots E_{K_{CC_1}}(E_{K_U}(m))\ldots))$; $sig_{CC_n}(sig_{CC_{n-1}}(\ldots sig_{CC_1}(u_m)\ldots)); u_m$

When the user wants to access his information, the decryption works as follows:

- $U \rightarrow CC_1 : u_m$
- $CC_1 \rightarrow U :$ $E_{K_{CC_{|t|}}}(\ldots E_{K_{CC_{|2|}}}(E_{K_{CC_1}}(m)\ldots)); u_m$
- RNG $\xrightarrow{p} U : K$
- $U \xrightarrow{p} CC_1 : E_K(E_{K_{CC_1}}(\ldots E_{K_{CC_{|t|}}}(m)\ldots)); u_m$
- $CC_1 \xrightarrow{p} CC_{|t|} : E_K(E_{K_{CC_{|2|}}}(\ldots E_{K_{CC_{|t|}}}(m)\ldots)); u_m$
- $CC_{|t|} \xrightarrow{p} CC_{|t-1|} :$ $E_K(E_{K_{CC_{|2|}}}(\ldots E_{K_{CC_{|t-2|}}}(m)\ldots)); u_m$
- ...
- $CC_{|2|} \xrightarrow{p} U : E_K(m); u_m$

Note that the user U does not store the key, he uses it only at the time of the encryption process;but since only t among n participants are needed to decipher the information, in the decryption protocol, the encrypted information can also be seen as the result of the encryption of the information by t different cloud providers. In the decryption protocol, the user realizes also an encryption with a fresh key K in order to guarantee the confidentiality of his information in regard to the last cloud provider who sends him the requested information. This fresh key is independent from the other keys K_{CC_i} and from K_U.

Even if a tenant can see the encrypted information, he does not have the key(s) to decrypt it. The same idea is relevant when there is a malicious entity within a cloud provider: no cloud provider has alone the secret key(s) needed to decrypt the secret information.

Since the transmitted data are encrypted, no one can understand the confidential information neither in the case where the data are transmitted between cloud provider's servers nor in the case where the data are transmitted between the cloud provider and the user.

The only security risk is a coalition of dishonest cloud providers which is relaxed with a high t relatively to n.

When dealing with n cloud providers, it suffices that at least $n-t+1$ cloud providers destroy their shares K_{CC_i} to make recovery of m infeasible from CC_1. Moreover there are no logfile links to the stored information m.

4 CONCLUSION

In this paper, we have considered the security issues related to the use of cloud computing infrastructures. We have proposed three protocols that allow to manage securely the needed cryptographic keys.

Our protocols are based on an $(t, n+1)$-threshold cryptosystem as well as on the principle of mutual cooperation between a set of cloud providers (therefore we can see it as a fair exchange of KMaaS between several cloud providers).

The main disadvantage of our third protocol is that each time when the user encrypts (respectively decrypts) a data, $n+1$ (respectively t) participants have to be involved. Moreover, each (encrypted) data has to be send to each participant. In other words, n, t

and the size of each sent data have to be carefully chosen depending on the capacity of the interCloud system. For example, for a fixed value of n, when t increases the number of communications between the cloud providers increases but the risk of coalition between cloud providers (in order to break the user confidentiality) is reduced.

However, the purposes of our third protocol are twofold. The client uses cryptographic keys without having to hold them locally. Moreover the cloud provider cannot retrieve client's information alone. Indeed the cloud providers retain only incomplete data.

We may assume the use of *true* random values when generating cryptographic keys in order to achieve the highest level of security in this matter. Moreover considering the specific context of cloud computing, given the significant available computing power as well as the potential massive use of random numbers, it is reasonable to target a higher level of security in order to be protected against attacks (Goldberg and Wagner, 1996; Woolley et al., 2008; Garfinkel and Rosenblum, 2005; Ristenpart and Yilek, 2010). Finally, we propose to use a fast random number generator for performance reason and a true random number generator for security reason.

The main issue is to know how to chose the t honest cloud providers to decrypt a data. There are not yet solution to find who are the dishonest cloud providers. Future work will focus on this interesting issue.

REFERENCES

K. kelly. http://www.kk.org/thetechnium/archives/2007/11/a_cloudbook_for.php.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley.

Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., and Morrow, M. (2009). Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In *ICIW*, pages 328–336. IEEE Computer Society.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In Hsu, C.-H., Yang, L. T., Park, J. H., and Yeo, S.-S., editors, *ICA3PP*, volume 6081 of *LNCS*, pages 13–31. Springer.

Catteddu, D. and Hogben, G. (2009). Cloud computing: benefits, risks and recommendations for information security. Technical report, ENISA.

Celesti, A., Tusa, F., Villari, M., and Puliafito, A. (2010). How to enhance cloud architectures to enable cross-federation. In *International Conference on Cloud Computing*, CLOUD '10, pages 337–345. IEEE Computer Society.

Garfinkel, T. and Rosenblum, M. (2005). When virtual is harder than real: security challenges in virtual machine based computing environments. In *Conference on Hot Topics in Operating Systems*, volume 10 of *HOTOS'05*, pages 20–20. USENIX Association.

Geater, J. (2011). Comment: Key management strategies in the cloud'. http://www.infosecurity-magazine.com/view/18818/comment-key-management-strategies-in-the-cloud.

Gellman, R. (2009). Privacy in the clouds : Risks to privacy and confidentiality from cloud. *Violence Against Women*, pages 1–26.

Goldberg, I. and Wagner, D. (1996). Randomness and the netscape browser. In *International Conference on Template Production*. Dr. Dobb's Journal.

Hogben, G. (July 2009). Privacy, security and identity in the cloud. ENISA.

informatique, S. (2010). Le livre blanc du cloud computing - tout ce que vous devez savoir sur l'informatique dans les nuage.

Kandukuri, B., Paturi, R., and Rakshit, A. (2009). Cloud security issues. In *International Conference on Services Computing*, SCC'09, pages 517–520. IEEE Computer Society.

Mather, T., Kumaraswamy, S., and Latif, S. (2009). *Cloud Security and Privacy: An Enterprise Perspective on Risk and Compliance*. O'Reilly.

Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. (2009). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Conference on Computer and Communications Security*, pages 199–212. ACM.

Ristenpart, T. and Yilek, S. (2010). When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *NDSS*. The Internet Society.

Rocha, F. and Correia, M. (2011). Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *International Conference on Dependable Systems and Networks Workshops*, DSNW'11, pages 129–134. IEEE Computer Society.

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

Sogeti (2009). Cloud computing - etat de l'art.

Squicciarini, A., Sundareswaran, S., and Lin, D. (2010). Preventing information leakage from indexing in the cloud. In *CLOUD*, pages 188–195. IEEE.

Urquhart, J. (7 Jan. 2009). The biggest cloud-computing issue of 2009 is trust. C-Net News.

Woolley, R., Murray, M., Dounin, M., and Ermilov, R. (2008). arc4random predictable sequence vulnerability. http://security.freebsd.org/advisories/FreeBSD-SA-08:11.arc4random.asc.