The Journal of Systems and Software 86 (2013) 2263–2268

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Cloud computing security: The scientific challenge, and a survey of solutions

Mark D. Ryan

School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK

ARTICLE INFO

Article history:
Received 24 September 2012
Accepted 7 December 2012
Available online 18 February 2013

Keywords:
Cloud computing
Homomorphic encryption
Trusted computing
Browser security

ABSTRACT

We briefly survey issues in cloud computing security. The fact that data are shared with the cloud service provider is identified as the core scientific problem that separates cloud computing security from other topics in computing security. We survey three current research directions, and evaluate them in terms of a running software-as-a-service example.

1. What is cloud computing security?

Cloud computing is the idea that data and programs can be stored centrally, in the cloud, and accessed any time from anywhere through thin clients and lightweight mobile devices. This brings many advantages, including data ubiquity, flexibility of access, and resilience. In many ways, it also enhances security: the cloud provider may be able to afford to invest in better and more up-to-date security technologies and practices than the data owner can. However, since cloud computing necessarily puts data outside of the control of the data owner, it inevitably introduces security issues too.

Cloud computing security concerns all the aspects of making cloud computing secure. Many of these aspects are not unique to the cloud setting: data is vulnerable to attack irrespective of where it is stored. Therefore, cloud computing security encompasses all the topics of computing security, including the design of security architectures, minimisation of attack surfaces, protection from malware, and enforcement of access control. But there are some aspects of cloud computing security that appear to be specific to that domain (Chen et al., 2010; Cloud Security Alliance, 2010; Christodorescu et al., 2009):

1 The cloud is typically a shared resource, and other sharers (called tenants) may be attackers.
2 Cloud-based data is usually intentionally widely accessible by potentially insecure protocols and APIs across public networks.
3 Data in the cloud is vulnerable to being lost (e.g., accidentally deleted) or incorrectly modified by the cloud provider.
4 Data in the cloud can be accessed by the cloud provider, its subcontractors and employees.

The technologies for addressing points 1–3 in the list above have existed for some time, and they are not specific to cloud computing security. The multi-tenancy aspect is addressed by deploying strong virtual machine managers and operating systems that ensure separation between processes; that was always their purpose. Securing data access across public networks, although essential to cloud computing, is in fact a challenge that pre-dates the subject, and there are already mature solutions including authentication protocols, authorisation frameworks, and encryption. Loss of data is addressed by careful backup policies, including remote backup. There are several techniques that can be used to detect incorrect modifications. Although deploying these solutions may be a significant management and engineering challenge, it is not a scientific challenge. From a scientific point of view, points 1–3 in the list above are solved problems.

Therefore, the genuinely unique challenge posed by cloud computing security boils down to just one thing: the data in the cloud can be accessed by the cloud provider. The cloud provider as a whole (or its employees individually) can deliberately or inadvertently disclose customers' data. Moreover, the cloud provider may have subcontractors (typically, a "software-as-a-service" provider will subcontract to an "infrastructure-as-a-service" provider), and the subcontractors may also have access to the data. This paper addresses the question of how a customer could secure its data from malicious or negligent cloud providers.

If the cloud's role is confined to storing the data on behalf of the owner, then the problem can be solved by encrypting the data, with the owner holding the keys. However, typically one wants the cloud provider to be able to do non-trivial computations with the

E-mail address: M.D.Ryan@cs.bham.ac.uk

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.12.025

data, and therefore the problem is in fact very hard to solve. Such computations may include searches, transformations, selections, and access control decisions. For example:

- The data is an email archive, and the cloud is expected to handle email in a way that depends on its content (e.g., spam filtering, topic classification, compliance with corporate disclosure policies).
- The data is documents and photographs, and the cloud is expected to enforce access rules that depend on the content.
- The data may be scientific data obtained by experiment and observation, and the computation may be to find patterns, or organise the data according to rules. Other examples in this category include data-intensive financial modelling or network traffic modelling.
- The data may be financial data, payroll data, human resource data, banking data, etc., and the computation is usual business processing.
- The data may be documents to review (job applications, research papers, business tenders) and the computation may be to support the review process (sending to reviewers, collating reviews, etc.).

1.1. A running example

To understand the importance of cloud-provider confidentiality, we detail the issues and considerations surrounding cloud-based conference management systems, which represent an example of these problems within the academic research community (Ryan, 2011). It is an interesting example, because it is small and specific, making it easier to explore the exact nature of the confidentiality problem and to think about solutions. Systems such as EasyChair and EDAS allow a conference chair or manager to create the conference account "in the cloud", and those systems handle all the necessary administration such as distribution of papers to programme committee (PC) members, collection and distribution of reviews and discussion, and production of emails to authors and reviewers and reports such as acceptance statistics and the conference programme. Because a cloud provider takes responsibility for the data across all conferences, the whole business of managing the server (including backups and security) is done by a third party different from the conference organisers or participants, and gains economy of scale. At the same time, accounts for authors and PC members exist already, and do not have to be managed on a per-conference basis.

The privacy concerns with cloud-computing-based conference management systems such as EDAS and EasyChair arise because the system administrators are custodians of a huge quantity of data about the submission and reviewing behaviour of thousands of researchers, aggregated across multiple conferences. This data could be deliberately or accidentally disclosed, with unwelcome consequences:

- Reviewer anonymity could be compromised, as well as the confidentiality of PC discussions.
- The acceptance success records could be identified, for individual researchers and groups, over a period of years.
- The aggregated reviewing profile (fair/unfair, thorough/scant, harsh/undiscerning, prompt/late, and so forth) of researchers could be disclosed.

The data could be abused by hiring or promotions committees, funding and award committees, and more generally by researchers choosing collaborators and associates. The mere existence of the data makes the system administrators vulnerable to bribery, coercion, and/or cracking attempts. If the administrators are also researchers, the data potentially puts them in situations of conflict of interest.

It is interesting to note that, as well as the confidentiality issues raised, there are some potential uses of the centralised data that might be considered beneficial. The data could be used to help detect or prevent fraud or other kinds of unwanted behaviour, for example, by identifying:

- Researchers who systematically unfairly accept each other's papers, or rivals who systematically reject each other's papers, or reviewers who reject a paper and later submit to another conference a paper with similar ideas.
- Undesirable submission patterns and behaviours by individual researchers (such as parallel or serial submissions of the same paper; repeated paper withdrawals after acceptance; and recurring content changes between submitted version and final version).

These examples show that it is desirable if the cloud can process the data in useful and interesting ways, but data confidentiality from the cloud provider remains paramount. The challenge of this example is to enable the cloud to process the data, while keeping the data confidential from the cloud provider.

1.2. Paper structure

In the next section, we review some of the research activities that address the core problem of cloud computing security, namely, the potential access by the cloud provider to customers' data. We briefly consider how well the proposed solution would work for the running example. (However, it is worth remembering that cloud computing is very diverse, and different examples may have quite different characteristics.)

2. Approaches to achieving confidentiality from the cloud provider

At present, this problem is handled mostly by legislation, contract, and good practice. The cloud provider does its best to have internal processes that restrict data access to as few employees as possible. Such arrangements rely substantially on data owners trusting the provider. A contract may exist to forbid the provider from disclosing the data to third parties, and it may be backed up by legislation that also aims to prevent the provider from acting unfairly. These approaches are difficult to scale up. If the cloud provider is very large, it will have subcontracting arrangements with other service providers, and they would have to be locked into the contract too. Also, the cloud provider may have numerous employees, and it may be very hard to vet them all to the standards wished by the data owner. Even cloud providers that have a huge reputation to protect, like Google, have had to sack employees for illegitimate access to customer data. It might become difficult for the owner to prove liability in the case of data breach, and even more difficult to take legal action and obtain compensation. Moreover, the international nature of cloud providers like Amazon and Google make it difficult to legislate their behaviour effectively.

Researchers are attempting to discover technological solutions which would give the data owner verifiable guarantees that their data remains confidential. As explained above, the requirement to perform in-cloud processing makes this complicated. In the remainder of this section, we describe a few of the approaches that are currently actively being researched.

2.1. Fully homomorphic encryption

As mentioned above, the data owner can encrypt the data before sending it to the cloud. With ordinary encryption, this prevents the cloud provider from operating on the data, limiting the cloud's role to simple storage. Homomorphic encryption overcomes this limitation: it is an encryption technique that allows a party that holds ciphertexts to perform certain operations on the ciphertexts, which mirror the corresponding operations on the plaintexts. In the case of simple homomorphic encryption, there is just one operation on the plaintext that has a corresponding operation on the ciphertext. For example, plain RSA has that property. Suppose ciphertext c_1 is the encryption under a public key pk of plaintext m_1, and c_2 is the encryption under the same key of m_2; that is

$$c_1 = enc(pk, m_1) \quad \text{and} \quad c_2 = enc(pk, m_2).$$

Then multiplying the ciphertexts results in something which, when decrypted, is identical to the result of multiplying the two plaintexts. That is, if sk is the decryption key corresponding to pk, then

$$m_1 \times m_2 = dec(sk, c_1 \times c_2).$$

This works with plain RSA, which is not used in practice because it is insecure. RSA with OAEP padding, which is secure and is used in practice, does not have this property. Nevertheless, there are other encryption schemes that are secure and do have the homomorphic property, such as Pallier encryption, and Elgamal encryption. In the case of plain RSA and Elgamal, multiplication of ciphertexts corresponds to multiplication of plaintexts; in Pallier, multiplication of ciphertexts corresponds to addition of plaintexts. Fully homomorphic encryption (Gentry, 2009) goes further: instead of there being just one plaintext operation that is mirrored by a ciphertext operation, every operation on plaintexts has a correspondent on ciphertexts, and therefore any program (seen as a sequence of operations) has a counterpart program that runs on ciphertexts. Suppose enc/dec is a fully homomorphic encryption scheme, and let p be a program that takes n inputs and produces an output. Then there exists a program p' such that if $c_i = enc(pk, m_i)$ are the encryptions of plaintexts m_i then

$$p(m_1, \ldots, m_n) = dec(sk, p'(pk, c_1, \ldots, cn)).$$

In other words, p' performs the operation on ciphertexts that produces a result which, when decrypted, is equal to the result of p on the plaintexts. (As a minor detail, note that, in general, p' may be given access to the public key as well.)

2.1.1. Discussion

Fully homomorphic encryption is a powerful idea and undoubtedly has a role to play in cloud computing. In principle, it means that the cloud provider can run the correspondent of any program the client wishes, while not obtaining access either to the argument data or the result data. When fully homomorphic encryption was first created in 2009, it was held up as a magic solution that would solve the big problem of cloud computing security (IBM). Unfortunately, however, this promise is unlikely to be realised. There are two fundamental reasons that appear to limit the contribution that fully homomorphic encryption can make to cloud security.

The first one is that the program p' necessarily outputs an encryption of the desired result, not the result itself. This means that the cloud cannot take action based on the result of the computation—it only has an encryption of the result, which it cannot decrypt. Naturally, randomised encryption is used (for otherwise there is no security at all), and therefore even the encryption of a binary output—true or false—tells the cloud nothing about the output itself. Suppose one wished to do email spam filtering in the cloud. We suppose that the cloud has a set of encrypted messages, and a program p which, when applied to a message, determines whether the message is spam or not. The cloud can run p' on a given encrypted message, resulting in an encrypted Boolean indicating whether the message is spam. Because of the mentioned limitation, the cloud cannot throw away the message if it is spam; instead, it can return the encrypted Boolean to the user, who can decrypt it and herself throw away the offending message. In other words, the cloud can do spam detection, but cannot do spam filtering. This may mean that the user is continually called upon as a decryption oracle, decrypting results to enable the cloud to take actions. Moreover, the user is required to evaluate each time whether it is appropriate to decrypt or not. If she decrypts carelessly, she may accidentally disclose confidential data to the cloud; yet, it may not be easy for her to determine whether a particular decryption is safe or not. It seems unclear whether a practical application of cloud computing could work satisfactorily in this way.

The second limitation of fully homomorphic encryption is its inefficiency. One has to fix in advance a limit on the size of the program p. Moreover, both the size of the ciphertext and the complexity of the encryption and decryption operations grow enormously with that limit on the size of p. In any non-trivial application, key sizes and the encryption of a single bit are measured in gigabytes. Storage is of order 10^{10} larger, and computations take much longer. One estimate is that Gentry's 2009 system gives one a 10^{12} slowdown. There is much current work to speed this up (e.g., Smart and Vercauteren, 2010), and orders of magnitude improvements have been obtained in the last few years. Even so, the inefficiencies of fully homomorphic encryption will severely limit its applications. Another, related, concern is that one is always dealing in encryptions of large quantities of data under a public key, something that is impractical even with non-homomorphic schemes. Researchers have shown that symmetric key AES decryption may itself be done as a data operation under a fully homomorphic public key scheme, but the result of such an AES decryption is still an encryption of data under a public key.

For these two reasons, fully homomorphic encryption is unlikely to gain widespread adoption. Although it is likely to find niche applications, it does not seem that it will become a solution to the cloud computing problem on a large scale.

2.1.2. The example

Fully homomorphic encryption can in principle solve the confidentiality problem of the conference management running example. To achieve this, we imagine the present situation with EasyChair, with the addition that all data sent to the cloud server is encrypted by a public key. All authors, reviewers, and chairs hold the secret key corresponding to this public key. The server performs the computations as at present (for example, managing and storing papers and reviews; enforcing access control), but the computations are done under the encryption. Data that the server sends to authors, reviewers and chairs are encrypted, and have to be decrypted by the recipient.

This solution is not attractive. As well as the two reasons detailed in the previous section which characterise fully homomorphic encryption in general, there are some more specific issues that arise in this example. First, the private part of the public key is held by a large number of individuals, and it is hard to keep such a key secret. One could try to address this by having a key for every conference, but this creates key distribution problems. Moreover, this approach (whether by global key or per-conference key) requires and relies on preventing the cloud provider from submitting a paper to the conference (since authors are entitled to the secret key). Another problem is that the cloud server cannot send mail to authors and reviewers, since their identities and the contents of the mail are all held in encrypted form. This might be solved by having a mail agent that knows the secret key and is able to receive encrypted mailing instructions and process them. However, the mail agent and the cloud server could conspire to decrypt everything.

It is quite likely that some of these problem-specific issues can be solved, but the solution remains unappealing for the bigger reasons.

2.2. Key translation in the browser

With this approach, data is encrypted before being uploaded to the cloud, and the data owners retain the keys. However, different parts of the data may be encrypted with different keys, and some of the clients participating in the service may perform "key translation" in order to allow data items to be forwarded to intended recipients. This does not allow arbitrary processing in the cloud as promised by fully homomorphic encryption, but it does allow several kinds of store-and-forward services. So far, a solution for a particular kind of software-as-a-service has been developed (Arapinis et al., 2012, under review), namely, services that support applications, evaluations, and decisions. The conference management running example of this paper is one such service; others include job application management, and public tender management (e.g., for civil construction). It is assumed that applicants, evaluators and decision makers interact through their web browsers with the cloud-based management system, to perform the usual task of uploading and downloading applications and evaluations. The cloud is responsible for fine-grained routing of information, in order to ensure that the right agents are equipped with the right data to perform their task. It is also responsible for enforcing access control, for example concerning conflicts of interest and to ensure that an evaluator does not see other evaluations of an application before writing her own. However, all the sensitive data is seen by the cloud only in encrypted form, and the cloud is not able to tell which applicants are being reviewed by which evaluators.

The technique relies on there being a process manager who oversees the data flow and is allowed access to all the data for a particular instance of the service (e.g., a particular recruitment exercise or conference organisation). When data is first introduced into the system by applicants when they upload applications, the data is encrypted with a data key, which is itself encrypted with a public key published and authenticated by the process manager. Thus, the cloud provider sees only encrypted data. Later, when the process manager needs to make this data available to reviewers, s/he downloads the data keys, decrypts them using the private key, and encrypts them again using a symmetric key that has been shared by an out-of-band protocol with the evaluators, and then uploads them again. This downloading, decryption, encryption and uploading can take place automatically by the web browser being used by the manager.

Not all data is withheld from the cloud provider. The provider needs access to some sensitive data, like the identity of the panel member in charge of evaluating a particular application, in order to implement the functional requirements of the protocol. Care in designing the protocol is needed to prevent known parties (such as applicants and evaluators) from being associated with each other by their handling of the same ciphertexts; for example, if an applicant uploads a particular ciphertext which is later downloaded by an evaluator, the cloud provider can potentially see an association between the applicant and the evaluator. To avoid this, the protocol makes use of re-randomised ciphertexts and randomised mixes.

2.2.1. Discussion

Key translation in the browser is so far restricted to a rather narrow class of applications, which roughly may be characterised as "store-and-forward". The cloud does not do real computation on the data, but still requires some access to data in order to facilitate information flow and enforce access control. We hypothesise that the class of applications can be expanded, perhaps including customer relationship management systems (such as salesforce.com), cloud-based finance and accounting services, and social networks, in which users share posts and status updates without enabling data to be mined by the cloud provider for profiling purposes. This remains a matter for further research.

2.2.2. The example

This technique was designed specifically for examples like the conference management one. Indeed, a prototype called ConfiChair has been implemented, with encouraging performance results (ConfiChair).

2.3. Hardware-anchored security

The final approach to achieving confidentiality from the cloud provider is based on special hardware on the cloud side. The idea is that the cloud provider is able to decrypt the data, but is able to offer guarantees about the circumstances in which it does that. Those guarantees will assure the data owners that the data is handled in accordance with their policy. Conceptually, it works something like this:

- The cloud provider has special hardware that allows it to store keys in a way that makes them accessible only to particular programs. We say such a key is bound to a program.
- The customer and the cloud provider agree a policy about how the data is to be manipulated. This policy is embodied in a program p.
- The client securely uploads to the cloud a key k, which is bound to p.
- The client uploads data encrypted with the key k to the cloud. The cloud can now run p, which can access k in order to manipulate the data. But the cloud cannot use any other program to manipulate the data, because a program different from p cannot access k.

As mentioned, special hardware is needed to realise this approach; it cannot be done in software alone. The key k has to be uploaded and stored in encrypted form, and it must be decrypted only in order to serve it to a running instance of p. While p is running, its memory and possibly its storage need to be protected from other programs running on the hardware.

The *trusted platform module* (TPM; Trusted Computing Group, 2007) is a hardware chip designed to enable this kind of binding between keys and programs, on commodity hardware. The TPM is an industry standard (and an ISO standard). It can work stand-alone by measuring the loaded software stack from boot time in a procedure called static root of trust. It works better with mechanisms for dynamically launching and measuring protected execution environments, such as Intel's *trusted execution technology* (TXT) or AMD's *secure virtual machine* (SVM); this procedure is called the dynamic root of trust.

2.3.1. Discussion and concrete instances

Putting these hardware components together in order to achieve a situation in which cloud providers can decrypt data only by programs that embody an agreed policy has turned out to be quite a challenge. Flicker (McCune et al., 2008) is a low-level architecture that achieves that, but only for small and short-lived programs running in impoverished environments (e.g., with basic input-output that excludes interaction with users or on the network). The main achievement of Flicker is to reduce the size of the *trusted computing base* (TCB)—that is, the program that has to be trusted (in addition to p)—to about 250 lines of code. TrustVisor (McCune et al., 2010) is another architecture by the same team, which is much more flexible, allowing the program p to include a full operating system (and therefore including software stacks for networking) running on a virtual machine, albeit at the cost of a larger TCB.

Excalibur (Santos et al., 2012) is a solution in this vein, with an even greater TCB but it provides a considerably richer and more usable platform based on Eucalyptus (Nurmi et al., 2008) which aims to be compatible with Amazon's EC2. In the Excalibur architecture, there is a monitor that acts as centralised authority and provides guarantees to the client. The monitor organises computations across a series of nodes. In Excalibur,

- The customer decides policy φ, and receives evidence of the monitor status in the form of key certificates and TPM status certificates called 'quotes', along with a public encryption key ek.
- The customer encrypts its data d and the policy φ, using ciphertext-policy attribute-based encryption (CPABE) (Bethencourt, 2007). This encrypted data is available to nodes too.
- The monitor receives TPM quotes from nodes, and maps them to a set of attribute–value pairs representing the security status of the node
- The monitor sends to each node the CPABE decryption keys corresponding to ek that match its security status. Because of the mechanism of CPABE, a monitor will be able to decrypt data only if its security status satisfies the policy φ specified by the client at encryption time.

Excalibur consists of 22 K lines of C code, including the monitor, the client-side library, the client-side demon for securing CPABE decryption keys, the management console, and the certificate toolkit. As mentioned, it provides a powerful and flexible environment for cloud-side processing with guarantees about confidentiality to the cloud. However, the size of the TCB makes it very difficult for the client to evaluate its security. To put this another way: the client needs to evaluate whether the entire software stack properly upholds the specified policy φ. Even if the software is open-source, it is very difficult to do that; there are some standard things one can look for, but establishing any property (such as security) of software is undecidable. The hardware (TPM and TXT) ensures that the software stack is the one being run, but it does not establish that it is secure. This phenomenon is sometimes called binary attestation, meaning that the guarantees are about what software binaries are running, rather than about the properties that are provided. Efforts at moving towards property-based attestation are currently being researched (Chen et al., 2006) and have partly been implemented in Excalibur, but the reliance on binary attestation is still a significant issue.

2.3.2. The example

There are several ways to leverage hardware-anchored security on the cloud side to achieve confidentiality for the conference management data. The most straightforward way is to use something like Excalibur (which provides secure infrastructure-as-a-service) to spin up a virtual machine (VM) with known software. This VM can be securely provisioned with a symmetric key (for disc storage) and the secret part of a public key (for TLS communication) by a conference chair at the time the conference account is created. The solution works, but has a few undesirable features. First, attestation is made to a huge software stack, and it is hard for conference chairs to evaluate it; moreover, the software stack is required to change frequently, due to security patches. Second, the two keys are held in memory for a long time, typically months, making them vulnerable to attacks on the running VM.

A better solution would be architected similarly to ConfiChair (Section 2.2), but with the key translation done by a small security kernel which is attested by the TPM. In principle, such a small kernel could be made using Flicker (McCune et al., 2008), which adds only a few hundred lines of TCB code. The total TCB size is then only one or two thousand lines of code, which is a manageable size. Keys are held in memory only for a few seconds, which is much more secure.

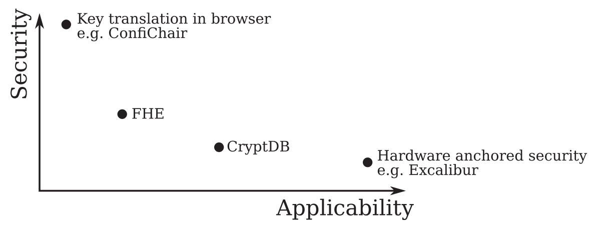

Fig. 1. The four approaches, impressionistically organised by their security and their applicability.

2.4. CryptDB: a weaker attacker model

CryptDB (Raluca et al., 2012) is a framework that allows query processing over an encrypted database. The database is stored and managed by the cloud provider, but data items are encrypted with keys that are not under the cloud provider's control. It works by executing SQL queries over the encrypted database using a collection of efficient SQL-aware encryption schemes. To achieve database operations such as equality checks and order comparisons, CryptDB uses encryption schemes that allow such comparisons to be made on ciphertexts. Such schemes have weak security, and represent CryptDB's philosophy of trading security and functionality with each other in an effort to reach a workable compromise. Another instance of this idea is CryptDB's technique of adjusting an onion-based encryption methodology according to queries observed at run time. Case studies for CryptDB include refereeing systems such as those described in Section 2.2.

CryptDB represents a weaker attacker model than the previous solutions discussed, because it assumes the existence of a trusted cloud-based application server and proxy. Nevertheless, CryptDB represents an interesting position on the trade-off between functionality and confidentiality from cloud providers.

3. Conclusions

The four approaches mentioned differ greatly in their applicability and the nature of the security guarantees they offer. The key translation approach offers perhaps the strongest guarantees, since strong cryptography can be used and the cloud provider never sees plain text data. However, it currently supports only a narrow set of store-and-forward applications, which is rather restricted. It was created with the conference management example in mind. CryptDB supports a wider set of applications, although it is still restricted to database searches. It uses conference management as an example application too. Its security can be considered a bit weaker than that of the key translation approach, because it deliberately uses weaker cryptography and trades security for data processability. Fully homomorphic encryption also appears to compromise security slightly, by using malleable encryption schemes, but probably not as much as CryptDB. However, the question of how applicable it is to real cloud computing problems is not clear. For example, it is not obvious how the conference management system could be securely implemented on top of fully homomorphic encryption, even if one ignores the scalability issues. Hardware-anchored security in the style of Excalibur is clearly the most versatile of the four approaches, but its security guarantees are of quite a different nature (and intuitively rather weaker) than the other three. Also, the nature of binary attestation (that is, attestation to a software stack rather than a policy) makes it hard to use in practice. These remarks are summarised in Fig. 1, which is intended to indicate an impression rather than to say anything very precise.

This space is fast moving, and we can expect to see many developments in the coming years.

References

Arapinis, M., Bursuc, S., Ryan, M., 2012. Privacy-supporting cloud computing: ConfiChair, a case study. In: Proceedings of the 1st Conference on Principles of Security and Trust (POST 2012).

Arapinis, M., Bursuc, S., Ryan, M., under review. Privacy-supporting cloud computing by in-browser key translation. Submitted to Journal of Computer Security.

Bethencourt, J., Sahai, A., Waters, B., 2007. Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334.

Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R., Stüble, C.,2006. A protocol for property-based attestation. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC'06). ACM, pp. 7–16.

Chen, Y., Paxson, V., Katz, R.H., 2010. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, Electrical Engineering and Computer Sciences, University of California at Berkeley.

Christodorescu, M., Sailer, R., Schales, D.L., Sgandurra, D., Zamboni, D., 2009. Cloud security is not (just) virtualization security: a short paper. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, pp. 97–102.

Cloud Security Alliance, 2010. Top threats to cloud computing v1.0. http://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

ConfiChair conference management system. http://www.confichair.org/

Gentry, C., 2009. Fully homomorphic encryption using ideal lattices. In: 41st ACM Symposium on Theory of Computing (STOC).

IBM researcher solves longstanding cryptographic challenge. Discovers method to fully process encrypted data without knowing its content; could greatly further data privacy and strengthen cloud computing security. http://www-03.ibm.com/press/us/en/pressrelease/27840.wss

McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Isozaki, H., 2008. Flicker: an execution infrastructure for TCB minimization. In: Proceedings of the ACM European Conference in Computer Systems (EuroSys), April.

McCune, J.M., Qu, N., Li, Y., Datta, A., Gligor, V.D., Perrig, A., 2010. Trustvisor: efficient TCB reduction and attestation. In: Proceedings of the IEEE Symposium on Security and Privacy, http://people.csail.mit.edu/costan/readings/oakland_papers/CMUCylab09003.pdf

Santos, N., Rodrigues, R., Gummadi, K.P., Saroiu, S., 2012. Policy-sealed data: a new abstraction for building trusted cloud services. In: USENIX Security.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D., 2008. Eucalyptus: a technical report on an elastic utility computing architecture linking your programs to useful systems. Technical report.

Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H., 2012. CryptDB: processing queries on an encrypted database. Communications of the ACM 55 (9), 103–111.

Ryan, M.D., 2011. Cloud computing privacy concerns on our doorstep. Communications of the ACM 54 (1), 36–38.

Smart, N.P., Vercauteren, F.,2010. Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Public Key Cryptography (PKC 2010). Springer.

Trusted Computing Group, 2007. TPM Specification version 1.2. Parts 1–3. http://www.trustedcomputinggroup.org/resources/tpm_main_specification

Mark Ryan is Professor of Computer Security at the University of Birmingham. His research focusses on design and analysis of security systems, being especially interested in systems that manage to balance requirements of individual privacy with accountability, and in formalising and verifying those properties. He has worked on the security and privacy properties of several applications, including electronic voting, mobile telephony, cloud computing, and trusted computing hardware.