Cloud Data Protection for the Masses

Abstract

Offering strong data protection to cloud users while enabling rich applications is a challenging task. We explore a new cloud platform architecture called Data Protection as a Service, which dramatically reduces the per-application development effort required to offer data protection, while still allowing rapid development and maintenance.

1 Introduction

Cloud computing promises lower costs, rapid scaling, easier maintenance, and services that are available anywhere, anytime. A key challenge in moving to the cloud is to ensure and build confidence that user data is handled securely in the cloud. A recent Microsoft survey [10] found that "...58% of the public and 86% of business leaders are excited about the possibilities of cloud computing. But, more than 90% of them are worried about security, availability, and privacy of their data as it rests in the cloud."

There is tension between user data protection and rich computation in the cloud. Users want to maintain control of their data, but also want to benefit from rich services provided by application developers using that data. At present, there is little platform-level support and standardization for verifiable data protection in the cloud. On the other hand, user data protection while enabling rich computation is challenging. It requires specialized expertise and a lot of resources to build, which may not be readily available to most application developers. We argue that it is highly valuable to build in data protection solutions at the platform layer: The platform can be a great place to achieve economy of scale for security, by amartizing the cost of maintaining expertise and building sophisticated security solutions across different applications and their developers.

1.1 Target Applications

There is a real danger in trying to "solve security and privacy for the cloud," because "the cloud" means too many different things to admit any one solution. To make any actionable statements, we must constrain ourselves to a particular domain.

We choose to focus on an important class of widely-used applications which includes email, personal financial management, social networks, and business applications such as word processors and spreadsheets. More precisely, we focus on deployments which meet the following criteria:

- applications that provide services to a large number of distinct end users, as opposed to bulk data processing or workflow management for a single entity;
- applications whose data model consists mostly of sharable data units, where all data objects have ACLs consisting of one or more end users (or may be designated as public);

- and developers who write applications to run on a separate computing platform—which encompasses the physical infrastructure, job scheduling, user authentication, and the base sofware environment—rather than implementing the platform themselves.

1.2 Data Protection and Usability Properties

A primary challenge in designing a platform-layer solution useful to many applications is allowing rapid development and maintenance. Overly rigid security will be as detrimental to cloud services' value as inadequate security. Developers do not want their security problems solved by losing their users! To ensure a practical solution, we consider goals relating to data protection as well as ease of development and maintenance.

Integrity. The user's private (including shared) data is stored faithfully, and will not be corrupted.

Privacy. The user's private data will not be leaked to any unauthorized person.

Access transparency. It should be possible to obtain a log of accesses to data indicating who or what performed each access.

Ease of verification. It should be possible to offer some level of transparency to the users, such that they can to some extent verify what platform or application code is running. Users may also wish to verify that their privacy policies have been strictly enforced by the cloud.

Rich computation. The platform allows most computations on sensitive user data, and can run those computations efficiently.

Development and maintenance support. Any developer faces a long list of challenges: bugs to find and fix, frequent software upgrades, continuous change of usage patterns, and users' demand for high performance. Any credible data protection approach must grapple with these issues, which are often overlooked in the literature on the topic.

1.3 Overview: Data Protection as a Service

Currently, users have to rely primarily on legal agreements and implied economic and reputational harms as a proxy for applications' trustworthiness. Ideally, we would like a robust technological solution as the base. To achieve this, we propose that the two most important things that a cloud platform could do are to:

- make it *easy* for developers to write performant, maintainable applications that protect user data in the cloud, thereby providing the same economies of scale to security and privacy as for computation and storage;
- enable *independent* verification both of the platform's operation and the runtime state of applications on it, so users (perhaps directly, but more likely via third-party auditors) can gain confidence that their data is being handled properly.

In the realm of data protection, people often view encryption as a kind of a silver bullet. In reality, encryption is just a tool—albeit a powerful one—to help achieve data protection properties, and not an end in itself. We will discuss two techniques that have received a lot of attention, full-disk encryption and computing on encrypted data, and show how they fall short of achieving our goals.

An approach in between those two, both in terms of key management and access-control granularity, is better suited for our target applications. We propose *Data Protection as a Service* [1]. Much as an operating system provides isolation between processes but substantial freedom inside a process, we aim for cloud platforms to offer transparently verifiable partitions for applications which compute on data units, while allowing broad computational latitude within those partitions. Our approach focuses on isolating data units from one another using trusted computing at runtime and cryptographic protections at rest, while centralizing access control for robust auditing. Crucially, our approach addresses directly the issues around rapid development and maintenance.

Our vision calls for cloud platform providers to offer Data Protection as a Service in addition to their existing hosting environment. Data Protection as a Service holds appeal for both users and application developers. Users would gain assurance about the security and privacy of their data while still benefiting from a rich spectrum of applications from diverse vendors. Developers could keep focus on innovation and agile development, offloading to the platform provider much of the burden of providing data protection to their customers. This service could be especially beneficial for small players in the market or those who do not have much in-house security expertise, allowing them to build user confidence much more quickly than they otherwise might.

2 Does Encryption Solve All Our Problems?

2.1 Full-disk Encryption and Computation on Encrypted Data

To motivate our ultimate proposal, we consider two different approaches to data privacy, full-disk encryption and computation on encrypted data.

Full-disk encryption (FDE) refers to encrypting entire physical disks with a symmetric key, often in disk firmware for simplicity and speed. Many standards call for encryption of data at rest, which FDE nominally fulfills. FDE is effective in protecting private data in certain scenarios such as stolen laptops and backup tapes. But does it fulfill our cloud data protection goals in the cloud, where physical theft is not the main threat?

At the other end of the spectrum, modern cryptography offers rich capabilities for computation on encrypted data which were not previously possible. Recently, the first realization of fully homomorphic encryption (FHE) was discovered by Craig Gentry [4]. FHE offers the promise of general computation on ciphertexts. That is, you can take *any* function on plaintext and transform it into an equivalent function on ciphertext: the server does all the real work, but does not know what the data it is computing on actually is. Naturally, this property gives strong privacy guarantees when computing on private data, but the question of its *practicality* for general cloud applications still remains.

2.2 A Comparison of FDE and FHE

Key management and trust. With FDE, the keys reside with the cloud platform, generally on or close to the physical drive. The end user for a cloud application is not involved in key management at all. While users' data is encrypted on disk, it exists in the clear in RAM whenever applications need to compute on the data. Consequently, FDE does not prevent online attacks—far more common in the cloud setting than physical attacks—from leaking the data to an unauthorized party.

In contrast, with FHE, untrusted applications cannot easily learn or leak users' data. FHE encryption keys are typically owned and managed by the users (although one can also imagine outsourcing key management to a third-party cloud provider), while applications compute on encrypted forms of users' data without actually "seeing" the data. This raises questions about how users can store their keys securely and reliably, especially in the presence of sharing. After all, the point of the cloud is not to have to maintain much local state.

Sharing. Collaboration is often cited as a "killer feature" for cloud applications. In our target scenario, fine-grained access control is necessary to allow a data owner to selectively share one or more data objects with other users.

If FDE is employed, users have to fully trust the cloud provider to enforce correct access control since the key granularity (the whole disk) does not line up with the granularity of access control (a single data unit).

If FHE is employed where the user (or a third-party cloud provider employed by the user) manages the encryption keys, it is not clear yet what will be the best way to provide access control. Specifically, to offer fine-grained encryption-based access control, key management may need to be performed on a per data object granularity, or over collections of data objects. However, using FHE, to support homomorphic operations across multiple encrypted objects, those objects must be encrypted under the same public key.

Aggregation. Many cloud applications require the ability to perform data mining over multiple users' data, for things such as spam filtering or computing aggregate statistics. With FDE, since users fully trust the cloud provider, performing such data aggregation is relatively easy.

Current FHE techniques do not readily allow computing on multiple users' data encrypted under different keys. Therefore, it is not clear yet how to support such data aggregation applications with FHE. Offline aggregation across users' data is also not possible. One could imagine addressing this by escrowing keys to the cloud provider, but at that point many of the benefits of FHE are lost, making its cost harder to justify.

Performance. Performance is an important factor. According to user studies conducted by Akamai [2], 49% of users will abandon a site or switch to a competitor after experiencing performance issues. Moreover, the need for speed is only increasing: in 2000, a typical user was willing to wait for 8 seconds for a webpage to load before navigating away, whereas by 2009, users were only willing to wait 3 seconds.

FDE is typically highly performant. When implemented in disk firmware, the symmetric encryption used can run at the full bandwidth of the disk, yielding effectively no slowdown. While significant advances have been made in improving the performance of FHE since Gentry's original proposal, there is still a long way to go before FHE can be made efficient enough to deploy at scale. In Gentry's estimation, implementing something like Google search would require roughly one trillion times more computation than the unmodified version [5].

Ease of development. Since FDE is hidden behind the abstraction of the physical disk, there is typically no impact on application development. In theory, FHE could also be relatively automatic; it works on an abstraction of the program as a circuit and transforms that circuit. In practice, performing this translation for arbitrary programs, especially when marshalling data, could be quite complex. At the least, programming tools would need to evolve dramatically.

FHE does not allow developers to input data-driven judgments into the development cycle. Specifically, FHE removes the ability of application developers to look at data, making it more

difficult to perform debugging, A/B testing, and application improvements.

Maintenance. Bugs are inevitable. Since availability is a primary goal, so is the need to debug quickly. When systems fail, it is often for some unforeseen reason, and people need to step in and manually take action. Figuring out the nature of the problem may require detecting unusual activity or understanding exactly what went wrong. In the FHE setting, this could be hard to do. If the application writer cannot inspect the state of the application meaningfully, debugging could be a real challenge. There will always be users who would rather their data be lost than be revealed for legitimate maintenance purposes, but this is a small minority, one that is often willing to pay the much greater costs of maintaining its own infrastructure.

2.3 Splitting the difference

As we have seen, FDE offers excellent performance and ease of development but does little to offer privacy at the granularity we require. FHE, on the other hand, pushes the privacy envelope in the other direction by removing data visibility entirely from the server and application developer. However, having a remote machine see and compute on sensitive data is not automatically a privacy violation. FHE's guarantees go beyond what is necessary to protect data by stated goals above, and in so doing incurs significant performance and development costs.

The approach we believe is better suited to our target applications is in a sense in between these two. It keeps the "natural" granularity of FHE by keying on units of sharable data, the performance of FDE by using symmetric encryption, and moves key management to a middle tier, the computing platform, to balance rapid development and easy maintenance with user-side verifiability.

3 A Way Forward: Data Protection as a Service

3.1 Key Principles

We propose the following key principles for Data Protection as a Service.

Lightweight confinement of user data. In an operating system, processes and files are the primary units of access control, and the OS provides suitable isolation for them. Applications are free to do what they like within these boundaries.

In a cloud setting, the unit of access control is typically a sharable piece of user data (e.g., a document in a collaborative editor). We would like some analogous confinement of that data, restricting its visibility only to authorized users and applications while allowing broad latitude for what operations are done on it. This can make writing secure systems *easier* for programmers, since confinement makes it more difficult for buggy code to leak data or compromised code to give unauthorized access to data. A malicious program may find different ways to exfiltrate data such as employing a side channel or covert channel; our higher priority here is to support benign developers, while making all applications and their actions on users' sensitive data more easily auditable to catch improper usage.

Clear audit trail for data access. One of the main concerns people and organizations have when putting data in the cloud is that they do not know what happens to it. Having a clear audit trail of when data is accessed, and by whom and what, can bolster confidence that data is being handled appropriately. Confinement can be effective for most normal user accesses, but

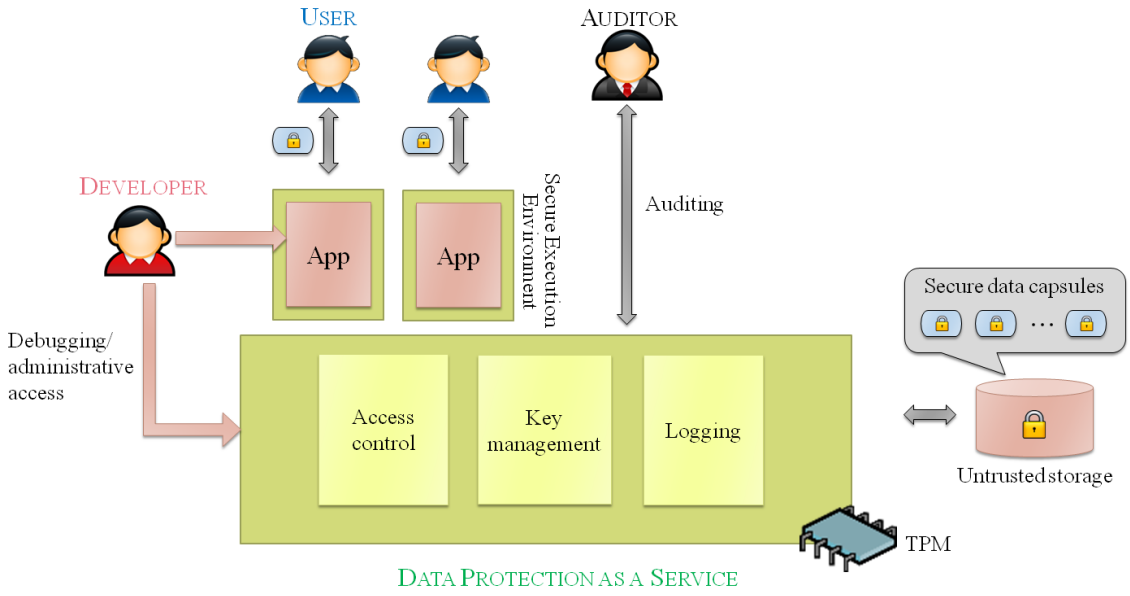

Figure 1: An Sample Architecture for Data Protection as a Service.

administrative access (e.g., debugging and analysis), which is outside the normal flow of user access and involves human administrators, can especially benefit from auditing.

Secure support for debugging, maintenance, and batch processing. These are essential for the proper functioning of virtually all useful applications. Bugs need to be fixed. Data need to be updated and migrated as schemas change. Offline computation is valuable for doing aggregation across users or for precomputation of expensive functions. To reduce the risk of unaudited back-door access, all these should be subject to the same authorization flows and platform-level checks as normal requests, albeit with a separate, appropriate policy.

Verifiable platform-level support. Support for confinement and auditing should be built into the platform in a verifiable way. This has many advantages: application developers do not all have to reinvent the wheel; the controls maintain independence from application code; third-party auditing and standards compliance are easier; and it allows for hardware support even in a virtualized environment. The cost of examining the platform is amortized across all its users; for a large-scale platform provider, this means significant economies of scale.

3.2 Design Space and A Sample Architecture

We will use a sample architecture [6], shown in Figure 1, to explore the design space of Data Protection as a Service. Here each server contains a Trusted Platform Module (TPM) providing secure and verifiable boot and execution.

3.2.1 Lightweight Confinement

Secure Data Capsules as data protection units A *Secure Data Capsule* (SDC) is an encrypted data unit packaged with its security policy, e.g., Access Control List (ACL). We will later describe how to use confinement and information-flow controls to enforce capsules' ACLs.

Runtime isolation We aim to support a rich spectrum of applications over users' data. To avoid unauthorized leakage of users' data in the presence of potentially buggy or compromised applications, we confine the execution of applications to mutually isolated environments, henceforth referred to as Secure Execution Environments (SEEs).

There are different levels of inter-SEE isolation we could impose; stronger isolation in general exacts a greater performance cost due to context switching and data marshalling. At one end, a SEE could be a virtual machine with an output channel back to the requesting user. In other words, the thread pool of a traditional server would be replaced with a pool of VMs or containers, whose data state is reset before being loaded with a new data unit. An intermediate approach would be to use OS process isolation, and even lighter-weight approach would be to use language-based features such as information flow controls [9], or capabilities [8]. Mechanisms like Caja for Javascript can be used to confine user data on the client side as well, though we do not include this as part of the platform.

In some cases, users' data may need to be exported from the system by design; the most common of these are using outside services or integrations. It is possible to mediate these exports via logging gateways, which provide a natural audit point. The same is true on the client side using mechanisms such as Caja or Dart.

Sharing and authorization It is a basic requirement of our target applications that units of data be sharable; that is why SDCs are bound to a full ACL versus a particular user. The key to enforcing those ACLs is that we can control the I/O channels available to SEEs. To confine data, we mandate that the data in an SDC is decrypted by the platform only for an SEE, and even then, only at an authorized user's request. The output may be funneled either directly to the user or to another SEE which provides a service; in either case, the channel is mediated by the platform.

The point is that a buggy SEE only exposes a single SDC. This is a big improvement over systems where a bug triggered by a particular malicious input allows access to arbitrary—that is, all—data.

ACL modifications, otherwise known as sharing or unsharing, are mediated too. A simple policy that the platform can enforce without having to know too much about the application is a transitive one: only currently-authorized users may modify the ACL. That is, the creating user is the first authorized user on the ACL, and at any time, an authorized user may modify the set by adding or removing users. Anonymous sharing, where possession of (say) a secret URL grants access to the data, is straightforward to support as well.

The platform itself need not understand granular, application-specific permissions; a simple, binary access-versus-no-access distinction goes a long way. The application may of course enforce any additional restrictions it requires on top of those provided by the platform. We do not require anything particular about the underlying storage service of the data unit.

This approach places two other requirements on the platform:

- the ability to perform user authentication, or at least have a trusted way to know who is logged in and accessing the service;

- to encrypt and MAC (respectively, decrypt and verify the MAC for) data on its way to and from storage, to remove the need to trust the storage service.

The first can be accomplished either with a proprietary approach or using open standards such as OpenID and OAuth. As for the second, since the platform mediates all the interaction, symmetric encryption suffices. With AES hardware units in commodity CPUs exceeding throughput of 1 GB/s/core, performance is unlikely to be a bottleneck for all but the most I/O-intensive applications. Once the data is loaded into the SEE, it need not be encrypted or decrypted again until being stored.

In this model, the application writer can offload much of the basic work for identity and ACL enforcement to the platform and get certain user-level guarantees for free. This alone makes it much easier for developers to reason about the security of their system since by default (without any authorized user present) the data is simply unavailable.

3.2.2 Data Access Auditing Support

Since the platform mediates all access to the data, authenticates users, and runs binaries, it knows what data is being accessed, by what user, and using which application. It can generate meaningful audit logs containing all these parameters and optionally incorporate additional information from the application layer.

There are four basic kinds of actions we can log:

- *Ordinary online data accesses* occur in response to external requests from users, and take place when a user is online and using an application.
- *Access control modification* by authorized users. Knowing the provenance of these changes can be helpful for forensics or diagnosing sharing problems.
- *Offline/batch access* to handle requests while users are offline (e.g., e-mail delivery), to compute aggregates, or to reorganize data such as during schema changes.
- *Administrative access* for maintenance operations such as debugging.

How detailed the logs are for each case can be decided on a case-by-case basis.

Given the ability to perform different types of audit, Data Protection as a Service can support third-party auditing services – which can be regarded as an additional player in this new ecosystem. These services can help end users make sense of how their data has been accessed and manipulated, and can also help them decide which services to trust. We anticipate that auditors will provide personalized services to particular end users, helping them understand how safe their data is with a particular service.

3.2.3 Authorization for Debugging, Maintenance, and Batch Access

While ordinary user access is governed by the ACL on the data, administrative access needs its own separate policy. That in turn can be audited to hold developers and administrators accountable. Each specific invocation of the administrative policy, because it may entail human access to data for, e.g., debugging or account recovery, should be logged and made available for auditing. The same kind of mechanism would be used for batch access, with perhaps different logging granularity;

to prevent misuse, the code for the batch process can be restricted only to an approved set. That might mean requiring controlled or quantifiable information release, such as differential privacy [3] or quantitative information flow [7].

3.2.4 Verifiability of the Platform

Having pushed these features into the platform, we can concentrate much of the auditing effort on it, sharing the benefits with all the applications running on top. Offline, the auditor can verify that the platform implements each data protection feature as promised. At runtime, the platform provider can use Trusted Computing technologies to attest to the particular software that it is running. Trusted Computing uses tamperproof hardware called Trusted Platform Module (TPM), as well as the virtualization and isolation features of modern processors, such as Intel VT or AMD-V. Trusted Computing allows for a dynamic root of trust; i.e., while the system is running, the CPU can enter a clean state, and a small secure kernel can be verified by the TPM, loaded, and executed. In particular, it can be verified by a third-party auditor, who amortizes the cost of this verification. The advantage to the developer here is to gain confidence that their code is really starting on the platform. Auditors too can provide more useful audit if it is clear what can and cannot be executed.

To allow for rapid pushes of new application binaries for scenarios such as bug fixes, the audit should include the procedures around updates and qualification of new releases rather than a simple "blessing" of a particular set of binaries whose lifetime is short.

For the application itself, getting from *verifiable* to *verified* is not easy; in a system with a lot of users, doing each pairwise verification independently is prohibitively expensive. That is where auditors come in. Certifications such as Statement on Auditing Standards Number 70 (SAS70) and others serve the important function of reducing the verification burden on both clients and service providers compared to doing pairwise examinations. These verifications in turn can be made easier by reusing the analysis of the platform.

3.3 Analysis against Our Data Protection Goals

We assume in this analysis that the platform has been verified to behave correctly with respect to code loading, authorization, and key management, and that there exists a runtime attestation to this effect (made possible by the TPM).

Data protection properties Privacy is ensured by the encryption at rest, and a combination of application confinement and information flow checking. Application confinement isolates faults and compromises within each SEE, while information flow checking ensures that any information flowing amongst SEEs, data capsules, and end users satisfy access control policies. Administrative accesses to data are controlled and audited to provide accountability.

Integrity of the data at rest can be obtained by cryptographic authentication of the data in storage, and by audit of the application code at runtime.

Ease of development Access controls, authorization, and auditing capability are common pain points for application developers. Getting these "for free" with the platform is a real improvement in ease of use, and we do not constrain the types of computation that can be performed within an SEE. Common maintenance tasks and batch processing are provided directly as first-class operations and

logged suitably. These too often require one-off work in the development process and can benefit from standardization.

4 Conclusion

As people's private data moves online, the need to secure it properly becomes ever more urgent. We can rely neither on the physical boundaries of the pre-Internet age, nor, with the cloud, the implicit protection afforded by data being distributed among individual users' machines. Implementing data protection for a cloud application is hard, requiring specialized skills and significant implementation effort.

The good news is that the same forces which are concentrating data in enormous datacenters are also the ones which will let us use our collective security expertise more effectively. Adding protections to a single cloud platform can immediately benefit hundreds of thousands of applications and, by extension, hundreds of millions of users.

Towards this goal, we have proposed a new paradigm for cloud computing, Data Protection as a Service, and discussed its key principles and design space. In this paradigm, the cloud platform not only provides the hardware and software stack as in today's cloud computing, but also dynamic data protection that protects users' data while enabling rich computation over them. We have focused here on a particular, albeit popular and privacy-sensitive, class of applications, and shown how to protect its data at the platform level. Many other types of applications also need solutions, and many practical questions still remain open.

We pose the following challenges:

- Can we standardize the technology across platforms, so switching between different providers is easy?
- How can we make migration for existing applications as easy as possible?
- How can we minimize the cost of application audits? What kind of audits are most important to build users' confidence?
- Can technologies such as Trusted Computing and code attestation be made scalable in the presence of constantly evolving software?
- How can we generalize the ideas here to other classes of applications?

In posing these questions, we hope to provoke thought and inspire future research and development in this important direction.

5 Acknowledgements

We gratefully acknowledge Krste Asanovic (UC Berkeley), Christoph Kern (Google), Petros Maniatis (Intel Labs), Prashanth Mohan (UC Berkeley), Charalampos Papamanthou (UC Berkeley), Alfred Spector (Google), Emil Stefanov (UC Berkeley), Mohit Tiwari (UC Berkeley), Nguyen Tran (New York University), and David Wagner (UC Berkeley) for valuable discussions and insightful feedback.

References

[1] http://www.mydatacontrol.com.

[2] The need for speed. http://www.technologyreview.com/files/54902/GoogleSpeed_charts.pdf.

[3] C. Dwork. The differential privacy frontier. In *TCC*, 2009.

[4] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC*, pages 169–178, 2009.

[5] A. Greenberg. IBM's Blindfolded Calculator. *Forbes*, June 2009. Appeared in the July 13, 2009 issue of Forbes magazine.

[6] P. Maniatis, D. Akhawe, K. Fall, E. Shi, S. McCamant, and D. Song. Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection. In *HotOS*, 2011.

[7] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *PLDI*, pages 193–205, 2008.

[8] M. S. Miller. Towards a Unified Approach to Access Control and Concurrency Control. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[9] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[10] L. Whitney. Microsoft Urges Laws to Boost Trust in the Cloud. http://news.cnet.com/8301-1009_3-10437844-83.html.