Cloud Data Protection for the Masses

Dawn Song, Elaine Shi, and Ian Fischer, *University of California, Berkeley*

Umesh Shankar, *Google*

The data-protection-as-a-service cloud platform architecture dramatically reduces the per-application development effort required to offer data protection while still allowing rapid development and maintenance.

Although cloud computing promises lower costs, rapid scaling, easier maintenance, and service availability anywhere, anytime, a key challenge is how to ensure and build confidence that the cloud can handle user data securely. A recent Microsoft survey found that "58 percent of the public and 86 percent of business leaders are excited about the possibilities of cloud computing. But more than 90 percent of them are worried about security, availability, and privacy of their data as it rests in the cloud."[1]

This tension makes sense: users want to maintain control of their data, but they also want to benefit from the rich services that application developers can provide using that data. So far, the cloud offers little platform-level support or standardization for user data protection beyond data encryption at rest, most likely because doing so is nontrivial. Protecting user data while enabling rich computation requires both specialized expertise and resources that might not be readily available to most application developers.

Building in data-protection solutions at the platform layer is an attractive option: the platform can achieve economies of scale by amortizing expertise costs and distributing sophisticated security solutions across different applications and their developers.

We propose a new cloud computing paradigm, *data protection as a service* (www.mydatacontrol.com). DPaaS is a suite of security primitives offered by a cloud platform, which enforces data security and privacy and offers evidence of privacy to data owners, even in the presence of potentially compromised or malicious applications.

SECURITY AND PRIVACY CHALLENGES

It's impossible to develop a single data-protection solution for the cloud because the term means too many different things. Any progress must first occur in a particular domain—accordingly, our work focuses on an important class of widely used applications that includes e-mail, personal financial management, social networks, and business tools such as word processors and spreadsheets. The following criteria define this class of applications:

- provide services to a large number of distinct end users, as opposed to bulk data processing or workflow management for a single entity;
- use a data model consisting mostly of sharable units, where all data objects have access control lists (ACLs) with one or more users; and
- developers could run the applications on a separate computing platform that encompasses the physical infrastructure, job scheduling, user authentication, and the base software environment, rather than implementing the platform themselves.

Overly rigid security is as detrimental to cloud service value as inadequate security. A primary challenge in

designing a platform-layer solution useful to many applications is ensuring that it enables rapid development and maintenance. To ensure a practical solution, we considered the following goals relating to data protection as well as ease of development and maintenance:

- *Integrity.* The user's stored data won't be corrupted.
- *Privacy.* Private data won't be leaked to any unauthorized entity.
- *Access transparency.* Logs will clearly indicate who or what accessed any data.
- *Ease of verification.* Users will be able to easily verify what platform or application code is running, as well as whether the cloud has strictly enforced their data's privacy policies.
- *Rich computation.* The platform will allow efficient, rich computations on sensitive user data.
- *Development and maintenance support.* Because they face a long list of challenges—bugs to find and fix, frequent software upgrades, continuous usage pattern changes, and user demand for high performance—developers will receive both development and maintenance support.

> **Currently, users must rely primarily on legal agreements and implied economic and reputational harm as a proxy for application trustworthiness.**

Any credible data protection approach must grapple with these issues, several of which are often overlooked in the literature.

DATA PROTECTION AS A SERVICE

Currently, users must rely primarily on legal agreements and implied economic and reputational harm as a proxy for application trustworthiness. As an alternative, a cloud platform could help achieve a robust technical solution by

- making it easy for developers to write maintainable applications that protect user data in the cloud, thereby providing the same economies of scale for security and privacy as for computation and storage; and
- enabling independent verification both of the platform's operation and the runtime state of applications on it, so users can gain confidence that their data is being handled properly.

Much as an operating system provides isolation between processes but allows substantial freedom inside a process, cloud platforms could offer transparently verifiable partitions for applications that compute on data units, while still allowing broad computational latitude within those partitions.

DPaaS enforces fine-grained access control policies on data units through application confinement and information flow checking. It employs cryptographic protections at rest and offers robust logging and auditing to provide accountability. Crucially, DPaaS also directly addresses the issues of rapid development and maintenance.

To truly support this vision, cloud platform providers would have to offer DPaaS in addition to their existing hosting environment, which could be especially beneficial for small companies or developers who don't have much in-house security expertise, helping them build user confidence much more quickly than they otherwise might.

WHAT ABOUT ENCRYPTION?

In the realm of data protection, developers often view encryption as a kind of a silver bullet, but in reality, it's just a tool—albeit a powerful one—to help achieve data protection properties. Although *full-disk encryption* (FDE) and computing on encrypted data have recently gained attention, these techniques have fallen short of answering all of the security and maintenance challenges mentioned earlier.

FDE encrypts entire physical disks with a symmetric key, often in disk firmware, for simplicity and speed. Although FDE is effective in protecting private data in certain scenarios such as stolen laptops and backup tapes, the concern is that it can't fulfill data protection goals in the cloud, where physical theft isn't the main threat.

At the other end of the spectrum, Craig Gentry recently proposed the first realization of *fully homomorphic encryption* (FHE),[2] which offers the promise of general computation on ciphertexts. Basically, any function in plaintext can be transformed into an equivalent function in ciphertext: the server does the real work, but it doesn't know the data it's computing. Naturally, this property gives strong privacy guarantees when computing on private data, but the question of its practicality for general cloud applications still remains.

FDE versus FHE

A comparison of FDE and FHE in the cloud computing setting reveals how these encryption techniques fall short of addressing the aforementioned security and maintenance challenges simultaneously.

Key management and trust. With FDE, the keys reside with the cloud platform, generally on or close to the physical drive: the cloud application user isn't involved in key management. While user data is encrypted on the physical disk, it is always accessible in the clear to any layer above it. Consequently, FDE doesn't prevent online attacks from

leaking the data to an unauthorized party, which is far more common in the cloud setting than physical attacks.

With FHE, untrusted applications can't easily learn or leak data. Users typically own and manage FHE encryption keys, while applications compute on encrypted forms of user 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 to avoid maintaining local state.

Sharing. Collaboration is often cited as a "killer feature" for cloud applications. Fine-grained access control is necessary to let a data owner selectively share one or more data objects with other users.

With FDE, users must fully trust the cloud provider to enforce correct access control because the key granularity (the whole disk) doesn't line up with access control granularity (a single data unit).

With FHE, because the user—or a third-party cloud provider employed by the user—manages the encryption keys, the best way of providing access control isn't clear yet. To offer fine-grained encryption-based access control, we might need to define key management on a per data object granularity basis or over collections of data objects. However, to support homomorphic operations across multiple encrypted objects, those objects must still be encrypted under the same public key.

Aggregation. Many cloud applications require performing data mining over multiple users' data for tasks such as spam filtering or computing aggregate statistics. Because users fully trust the cloud provider, performing such data aggregation is relatively easy with FDE.

Current FHE techniques don't readily allow computing on multiple users' data encrypted under different keys. Therefore, it isn't clear yet how to support such data aggregation applications with FHE; similarly, offline aggregation across users' data isn't possible. One solution might be to escrow keys to the cloud provider, but that would eliminate many of FHE's benefits, making its cost harder to justify.

Performance. According to a recent survey, 49 percent of users abandon a site or switch to a competitor after experiencing performance issues.[3] And the need for speed is only increasing: in 2000, a typical user was willing to wait 8 seconds for a webpage to load before navigating away; by 2009, that number dropped to 3 seconds.

When FDE is implemented in disk firmware, its symmetric encryption can run at the disk's full bandwidth, effectively avoiding a slowdown. Although researchers have made significant advances in improving FHE's performance since Gentry's original proposal, it has a long way to go before becoming efficient enough to deploy at scale. In Gentry's estimation, implementing something like a Google search with FHE would require roughly 1 trillion times more computation than the one without FHE.[4]

Ease of development. Because FDE is hidden behind an abstraction of the physical disk, it typically has 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, however, performing this translation for arbitrary programs—especially when marshaling data—could be quite complex. At a minimum, programming tools would need to evolve dramatically.

FHE doesn't allow developers to input data-driven judgments into the development cycle. Specifically, application developers can't look at the data, making debugging, A/B testing, and application improvements more difficult.

Maintenance. Bugs are inevitable. However, availability is a primary cloud goal, so the need to debug quickly is a top priority. Systems often fail for some unforeseen reason, requiring someone to step in and manually take action. Determining the nature of the problem might require detecting unusual activity or understanding exactly what went wrong, which isn't easy with FHE. If the application writer can't inspect application state meaningfully, debugging could be a real challenge.

> **The DPaaS approach moves key management and access control to a middle tier—the computing platform—to balance rapid development and easy maintenance with user-side verifiability.**

Splitting the difference

Although FDE offers excellent performance and ease of development, it does little to protect privacy at the required granularity. FHE, on the other hand, pushes the privacy envelope in the other direction by removing data visibility entirely from both the server and application developer. However, having a remote machine see and compute on sensitive data isn't automatically a privacy violation. FHE's guarantees go beyond what's necessary to protect data, and in so doing, it incurs significant performance and development costs.

We believe the DPaaS approach is better suited for the target applications because it falls between the two. It keeps the "natural" granularity of FHE by keying on units of sharable data and maintains the performance of FDE by using symmetric encryption. It moves key management and access control to a middle tier—the computing platform—to balance rapid development and easy maintenance with user-side verifiability.

A WAY FORWARD

In an OS, processes and files are the primary units of access control, and the OS provides suitable isolation for
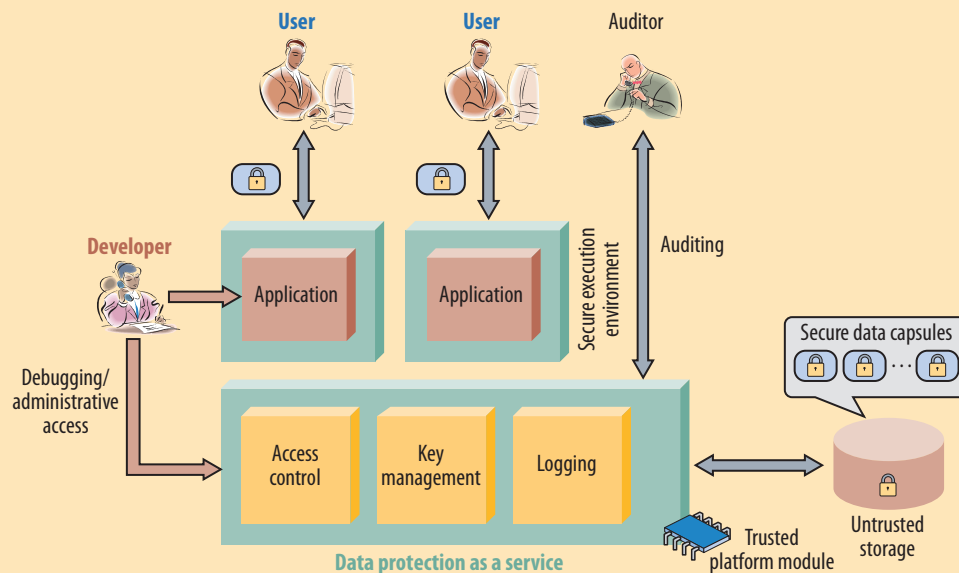
Figure 1. Sample architecture for data protection as a service illustrates how it's possible to integrate various technologies, such as application confinement, encryption, logging, code attestation, and information flow checking to realize DPaaS.

them. Applications can do what they like within these boundaries.

In a cloud setting, the unit of access control is typically a sharable piece of user data—for example, a document in a collaborative editor. Ideally, the system offers 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 because confinement makes it more difficult for buggy code to leak data or for compromised code to grant unauthorized access to data. A malicious program might find different ways to exfiltrate data, such as employing a side channel or covert channel, but the 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.

One of the main concerns people and organizations have about putting data in the cloud is that they don't know what happens to it. Having a clear audit trail of when data is accessed—and by whom or what—bolsters confidence that data is being handled appropriately. Confinement can be effective for most normal user accesses, but administrative access that's outside the normal flow of user access and involves human administrators (for example, for debugging and analysis) can especially benefit from auditing.

Verifiable platform support

Bugs need to be fixed. Data needs to be updated and migrated as schemas change. Offline computation is valuable for data aggregation across users or for precomputation of expensive functions. To reduce the risk of unaudited backdoor access, all these functions should be subject to the same authorization flows and platform-level checks as normal requests, albeit with a separate, appropriate policy.

Platform providers should build support for confinement and auditing into the platform in a verifiable way. This approval has many advantages:

- application developers don't have to reinvent the wheel;
- application code is independent of ACL enforcement;
- third-party auditing and standards compliance are easier; and
- the verifiable platform extends to virtualized environments built atop it.

Finally, the cost of examining the platform is amortized across all its users, which means significant economies of scale for a large-scale platform provider.

Design space and a sample architecture

Figure 1 illustrates an example architecture for exploring the DPaaS design space.[5] Here, each server contains a *trusted platform module* (TPM) to provide secure and verifiable boot and dynamic root of trust. This example architecture demonstrates at a high level how it's potentially possible to combine various technologies such as application confinement, encryption, logging, code attestation, and information flow checking to realize DPaaS.

Confinement. A *secure data capsule* (SDC) is an encrypted data unit packaged with its security policy. For example, an SDC might encompass a sharable document or a photo album along with its ACL. The platform can use confinement and information-flow controls to enforce capsules' ACLs.

To avoid unauthorized leakage of user data in the presence of potentially buggy or compromised applications, DPaaS confines the execution of applications to mutually isolated *secure execution environments* (SEEs).

Inter-SEE isolation has different levels, but stronger isolation generally exacts a greater performance cost due to context switching and data marshaling. At one end, a SEE could be a virtual machine with an output channel back to the requesting user. For performance reasons, it's possible to have a pool of VMs or containers in which the data state is reset before being loaded with a new data unit—similar to how a thread pool works in a traditional server. A more lightweight approach would be to use OS process isolation; an even lighter-weight approach would be to use language-based features such as information-flow controls[6] or capabilities.[7] We can use mechanisms such as Caja for JavaScript to confine user data on the client side as well, although we don't include that option as part of the platform.

In some cases, applications need to call outside services or APIs provided by third-party websites—for example, the Google Maps API. An application might need to export users' data to outside services in this process. Users can explicitly define privacy policies to allow or disallow exporting SDCs to such third-party services, and DPaaS can enforce these policies. Additionally, DPaaS can log all instances where data is exported, and an auditor can later inspect these logs and detect any misuse a posteriori.

Because our target applications have a basic requirement of sharable data units, DPaaS supports ACLs on SDCs. The key to enforcing those ACLs is to control the I/O channels available to the SEEs. To confine data, the platform decrypts the SDC's data only in a SEE in compliance with the SDC's security policy. A SEE can funnel the output either directly to the user or to another SEE that provides a service; in either case, the platform mediates the channel. A buggy SEE only exposes a single SDC, an improvement over systems in which malicious input triggers a bug that allows access to all data.

The platform also mediates ACL modifications, otherwise known as sharing or unsharing. A simple policy that the platform can enforce without having to know too much about the application is transitive: only currently authorized users can modify the ACL. For example, the creator is the first owner of a data unit, and at any time, any user with the owner status can add or revoke other authorized users. The support of anonymous sharing, in which possession of, say, a secret URL grants access to data, is also straightforward.

The platform itself doesn't need to understand granular, application-specific permissions; a simple, binary access-versus-no-access distinction goes a long way. The application can, of course, enforce any additional restrictions it requires on top of those the platform provides. There are no particular requirements for the data unit's underlying storage service.

The DPaaS approach places two additional requirements on the platform:

- it must be able to perform user authentication, or at least have a trusted way to know who's logged in and accessing the service; and
- it must rely on encryption and authenticated data store techniques to remove the need to trust the storage service.

> **Because DPaaS mediates all data access, authenticates users, and runs binaries, it knows what data is accessed by what user, and with which application.**

DPaaS can accomplish user authentication either with a proprietary approach or using open standards such as OpenID and OAuth. Because the platform mediates all interactions, symmetric encryption suffices. With AES hardware units in commodity CPUs exceeding throughput of 1 Gbyte/second/core, performance is unlikely to be a bottleneck for all but the most I/O-intensive applications. Once the system loads the data into the SEE, it doesn't need to be encrypted or decrypted again until storage.

In this model, the application 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 system security because, by default (without any authorized user present), the data is simply unavailable.

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

DPaaS can log four basic kinds of actions:

- ordinary online data accesses that occur in response to external user requests when a user is online and operating an application;
- access control modification by authorized users, the provenance of which can assist in forensics or problem diagnosis;

- offline/batch access to handle requests while users are offline (for example, e-mail delivery) to compute aggregates or to reorganize data such as during schema changes; and
- administrative access for maintenance operations such as debugging.

Users or developers can decide how detailed the logs are on a case-by-case basis.

Given its ability to perform different types of audit, DPaaS can also support third-party auditing services, thus helping users understand how their data has been accessed and manipulated, and which services to trust. We anticipate that auditors will provide personalized services to particular users, helping them determine how safe their data is with a particular service.

> **The same forces concentrating data in enormous datacenters will also aid in using collective security expertise more effectively.**

The ACL governs ordinary user access, but administrative access requires its own separate policy, which in turn can be audited to hold developers and administrators accountable. Because each specific invocation of the administrative policy might entail human access to data, it should be logged and made available for auditing. The same kind of mechanism could handle batch access, perhaps with different logging granularity. To prevent misuse, the platform can restrict batch processes to only an approved set of programs, for example, requiring the programs to have controlled or quantifiable information release, such as differential privacy[8] or quantitative information flow.[9]

Platform verifiability. The DPaaS approach provides logging and auditing at the platform level, sharing the benefits with all 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* (TC) technologies to attest to the particular software that's running. TC uses the tamperproof TPM as well as the virtualization and isolation features of modern processors, such as Intel VT or AMDV.

TC also allows for a dynamic root of trust—while the system runs, the CPU can enter a clean state, and the TPM can verify, load, and execute a *trusted computing base* (TCB), which is responsible for security-critical functionalities such as isolation enforcement, key management, access control, and logging. Moreover, a third-party auditor can verify the code of the TCB that has been loaded onto the cloud platform. In this way, users and developers can gain confidence that the applications are indeed running on the correct TCB, and consequently trust the security guarantees and the audit logs the TCB provides.

One challenge in code attestation is how to establish a set of acceptable binaries in the presence of rapid software updates such as bug fixes and new features. One potential way is to log the history of software updates and perform verification a posteriori.

For the application itself, getting from verifiable to verified isn't easy; in a system with a lot of users, doing all-pairs verification is prohibitively expensive. This 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 pairwise examinations. Since applications have the data-protection piece in common from the platforms, the application verifications in turn can be simpler than they otherwise would have been.

Achieving data protection goals

We assume in the analysis that the platform behaves correctly with respect to code loading, authorization, and key management, and that the TPM facilitates a runtime attestation to this effect.

DPaas uses a combination of encryption at rest, application confinement, information flow checking, and auditing to ensure the security and privacy of users' data. Application confinement isolates faults and compromises within each SEE, while information flow checking ensures that any information flowing among SEEs, data capsules, and users satisfies access-control policies. Controlling and auditing administrative accesses to data provides accountability. DPaaS can guarantee the integrity of the data at rest via cryptographic authentication of the data in storage and by auditing the application code at runtime.

Access controls, authorization, and auditing capability are common challenges for application developers. Incorporating these features within the platform is a significant improvement in terms of ease of use, and it doesn't constrain the types of computation that can be performed within a SEE. The platform logs common maintenance and batch processing tasks to provide accountability. These tasks too often require one-off work in the development process and can benefit from standardization.

A s private data moves online, the need to secure it properly becomes increasingly urgent. The good news is that the same forces concentrating data in enormous datacenters will also aid in using 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.

While we have focused here on a particular, albeit popular and privacy-sensitive, class of applications, many other applications also need solutions, and many practical questions still remain open:

- Can we standardize technology across platforms to facilitate switching among providers?
- How can we make migration to the DPaaS cloud as easy as possible for existing applications?
- How can we minimize the cost of application audits?
- What kinds of audits are most important for building user confidence?
- Can technologies such as TC and code attestation be made scalable in the presence of constantly evolving software?
- How can we generalize the ideas presented 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. ▣

Acknowledgments

References

1. C. Dwork, "The Differential Privacy Frontier Extended Abstract," *Proc. 6th Theory of Cryptography Conf.* (TCC 09), LNCS 5444, Springer, 2009, pp. 496-502.
2. C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," *Proc. 41st Ann. ACM Symp. Theory Computing* (STOC 09), ACM, 2009, pp. 169-178.
3. E. Naone, "The Slow-Motion Internet," *Technology Rev.*, Mar./Apr. 2011; www.technologyreview.com/files/54902/GoogleSpeed_charts.pdf.
4. A. Greenberg, "IBM's Blindfolded Calculator," *Forbes*, 13 July 2009; www.forbes.com/forbes/2009/0713/breakthroughs-privacy-super-secret-encryption.html.
5. P. Maniatis et al., "Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection," *Proc. 13th Usenix Conf. Hot Topics in Operating Systems* (HotOS 11), Usenix, 2011; www.usenix.org/events/hotos11/tech/final_files/ManiatisAkhawe.pdf.
6. S. McCamant and M.D. Ernst, "Quantitative Information Flow as Network Flow Capacity," *Proc. 2008 ACM SIGPLAN Conf. Programming Language Design and Implementation* (PLDI 08), ACM, 2008, pp. 193-205.
7. M.S. Miller, "Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control," PhD dissertation, Dept. of Philosophy, Johns Hopkins Univ., 2006.
8. A. Sabelfeld and A.C. Myers, "Language-Based Information-Flow Security," *IEEE J. Selected Areas Comm.*, Jan. 2003, pp. 5-19.
9. L. Whitney, "Microsoft Urges Laws to Boost Trust in the Cloud," *CNET News*, 20 Jan. 2010; http://news.cnet.com/8301-1009_3-10437844-83.html.
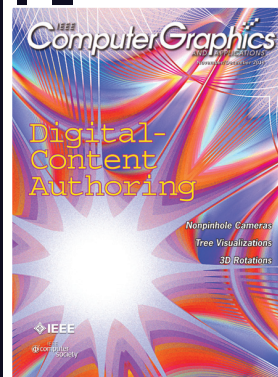
Dawn Song *is an associate professor of computer science at the University of California, Berkeley. Her research interests include security and privacy. Song received a PhD in computer science from the University of California, Berkeley. She is a member of IEEE and ACM. Contact her at dawnsong@cs.berkeley.edu.*

Elaine Shi *is a project research scientist at the University of California, Berkeley. Her research interests include security, privacy, applied cryptography, and data mining. Shi received a PhD in computer science from Carnegie Mellon University. Contact her at runting@cs.cmu.edu.*

Ian Fischer *is a PhD student in electrical engineering and computer science at the University of California, Berkeley. His research interests include security and privacy, security usability, and security applications of machine learning. Fischer received an MS in computer science from Harvard University. Contact him at ian.fischer@eecs.berkeley.edu.*

Umesh Shankar *is a senior staff engineer at Google. His research interests include data protection, usable security and privacy, and online reputation. Shankar received a PhD in computer science from the University of California, Berkeley. He is a member of IEEE and ACM. Contact him at ushankar@google.com.*