Bulletin of the Technical Committee on

Data Engineering

December 2012 Vol. 35 No. 4

IEEE Computer Society

Letters

Special Issue on Security and Privacy in the Cloud

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

Twenty Years at the Bulletin

It is hard for me to believe that 20 years have gone by since I took up the task of being Bulletin editor. It surely has not seemed that long– a sure sign that I have enjoyed the job. Over the years, the Bulletin has changed in format but not in purpose. In format, the Bulletin has gone from being a purely paper publication to one with both paper and a web presence, to finally a purely electronic web form. The primary format for each issue is now pdf, with a web table of contents. These all seemed new and interesting at the time, but have now simply become "the way things are".

One thing that hasn't changed is the Bulletin mission, which is to publish issues focused on a particular topic, containing early papers, and bringing together both academic and industrial authors. It is this mission that has kept me engaged for the past twenty years. I hope you all have enjoyed participating in this endeavor, whether as editors, authors, readers, or a combination of all these roles. And thank you all for the role you have permitted me to play. It has not all been fun, but it has been deeply satisfying.

TCDE Chair Election Results

I want to congratulate Kyu-Young Whang, who this fall was elected as the Chair of the Technical Committee on Data Engineering. Kyu-Young has a distinguished career as a database researcher and is an "eminence grise" of the Korean database community. Kyu-Young also has extensive experience in professional organizations, including both the ICDE Steering Committee and the TCDE Executive Committee. You can read Kyu-Young's introductory TC Chair letter on page 2. I wish Kyu-Young the very best as he starts his tenure as chair.

The Current Issue

I believe that "economics rules". That is, a low-priced alternative, assuming it is in most respects comparable to a high-priced alternative and with a large cost differential, will win the market. An historical example is PC-based servers, which were substantially lower in cost than either mainframe or mini-computer servers, while being "roughly comparable" in other respects. That kind of cost differential now applies when comparing servers a customer hosts himself vs servers in the cloud.

The question then is whether cloud-based servers can be made "roughly comparable" in other respects to servers on customer premises. This is where security and privacy enter the picture. On-premises servers have at least the illusion of being secure, in part because of lockable doors and trusted staff. In the cloud, things are much murkier. It would seem that it is the cloud provider whose doors need to be locked and whose staff needs to be trusted. So a customer has much less control of these aspects and is correct in proceeding carefully.

The current issue of the bulletin addresses exactly this topic. Sharad Mehrotra, as issue editor, has brought together a cross-section of papers in exactly the area of security and privacy, focused on how to provide them in the cloud. This is a technical challenge, and one not fully faced in the past. Hence it is both a great research area and a very important technical area and challenge. And there is money riding on the outcome!

This issue brings together a diversity of approaches to security and privacy. And while this is clearly not the last word on these subjects, it can serve as a great overview of the area and a very encouraging sign that progress is being made. I want to thank Sharad for his efforts in successfully bringing to the issue a very broad collection of approaches in an exciting and challenging area.

David Lomet
Microsoft Corporation

Letter from the New TCDE Chair

It is an honor to be elected Chair of Technical Committee on Data Engineering (TCDE), and I thank all the TCDE members for the trust and support. An IEEE organization, TCDE leads and serves the worldŠs database research community focusing on various aspects of data management and systems. TCDE involves various activities including but not limited to the following.

(1) We sponsor IEEE ICDE, a premier database conference, which now has a 28-year history and has grown to be strong and authoritative in our field. Since its inception, a lot of people contributed to making it a premier venue in our field, and we are committed to making it ever stronger to better serve our research community. I will work very closely with the ICDE steering committee to strengthen this flagship conference, in particular, to increase the visibility of and citations to the papers published in the ICDE proceedings. We also sponsor/co-sponsor other conferences/workshops. We have recently been co-sponsoring Mobile Data Management (MDM) and Social Network Analytics and Mining (ASONAM). I will encourage new applications for co-sponsorship in diverse areas to expand the coverage of TCDE.

(2) We publish IEEE Data Engineering Bulletin, a quarterly publication that has a 35-year history and has grown to be a highly respected research vehicle. This achievement was made largely due to the dedicated effort of David Lomet, the current Editor-in-Chief (EIC), as well as other earlier EICŠs. We will continue the Bulletin with David Lomet.

(3) We support working groups to promote research and activities in specific areas of interests. Currently, we have two working groups: Self-Managing Database Systems and Data Management in the Cloud. I will encourage new working groups in timely sub-areas of data management while continuing to provide good support to the existing ones.

(4) For many years TCDE has been leading the database community together with other representative academic societies such as ACM SIGMOD and VLDB Endowment. I will seek active cooperation with these sister societies on various issues arising from our community to derive as much synergetic effect as we can.

Obviously, all the efforts above will be in line with the excellent framework David Lomet and my other predecessors have built and accumulated. I sincerely thank them for their dedicated efforts towards this achievement.

Kyu-Young Whang
KAIST

Letter from the Special Issue Editor

Fuelled by the advances virtualization and high-speed network technologies, cloud computing is emerging as a dominant computing paradigm for the future. Almost all technology assessment groups such as Forrester and Gartner project very aggressive growth in cloud computing over the next decade or two. Cloud computing can roughly be summarized as "X as a service" where X could be a virtualized infrastructure (e.g., computing and/or storage), a platform (e.g., OS, programming language execution environment, databases, web servers), software applications (e.g., Google apps), a service, or a test environment, etc. A distinguishing aspect of cloud computing is the utility computing model (aka pay-as-you-go model) where users get billed for the computers, storage, or any resources based on their usage with no up-front costs of purchasing the hardware/software or of managing the IT infrastructure. The cloud provides an illusion of limitless resources which one can tap into in times of need, limited only by the amount one wishes to spend on renting the resources. Cloud computing is by no means a novel/new thought. In a recent tutorial on cloud computing at EDBT 2012, Amr Abbadi & Divy Aggrawal pointed out one of the early references to cloud computing in a speech by John McCarthy at the MIT centennial in 1961 where he states "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility Ě The computer utility could become the basis of a new and important industry". While McCarthy was possibly 40-50 years ahead in his projection, there is no doubt that the spirit of his statement is finally coming to fruition.

A sales pitch for cloud computing emphasizing its key characteristics could look something as follows: use as much as **your** needs dictate; pay **only** for what you use; don't **worry** about hiring staff to manage any system administration issues such as **loss of** data due to failures; and have better **control** over your IT investment (no up-front costs, cheaper due to economies of scale). Hidden in the sales pitch is what is, perhaps, the largest challenge facing cloud computing - **"your only worry is loss of control"**. Indeed, perception of loss of control over ones resources, whether it be infrastructure, software, or data, has been identified by many pundits as the important and immediate challenge facing cloud computing. The key operative issue here is the notion of trust. Loss of control, in itself, is not as much of an issue if clients/users could fully trust the service provider. In a world where service providers could be located anywhere, under varying legal jurisdictions; where privacy and confidentiality of ones data is subject to policies and laws that are at best (or under some circumstances) ambiguous; where policy compliance is virtually impossible to check, and the threat of "insider attacks" is very real - trust is a difficult property to achieve. Loss of control over resources (by migrating to the cloud) coupled with lack of trust (in the service provider) poses numerous concerns about data integrity, availability, security, privacy and confidentiality to name a few. Whether or not one migrates to the cloud depends upon how one balances the perceived risks (due to loss of control) with the benefits (the cloud offers) and this, in turn, depends upon the end users and their needs. It is perhaps fair to state that, given the widespread adoption of services such as email (e.g., Gmail), document management (e.g., Google Drive) on the internet, the end-users have already decided that the benefits outweigh the risks. The major question facing the cloud computing market is the extent to which small/medium/large organizations (including the government) - i.e. the "real" paying customers - will adopt cloud computing solutions. The answer to this question depends upon the perceived risks of migrating to the cloud and the organizations' risk-tolerance. The research community can significantly facilitate such a migration by developing technological solutions that help alleviate these risks.

This issue of the Data Engineering Bulletin focuses on the privacy and security aspects of outsourcing data to the cloud. The bulletin presents 11 papers by leading researchers who are exploring issues relevant to security, privacy, and confidentiality in cloud computing from different perspectives.

The bulletin starts with a paper by Montjoye, Wang, and Pentland that lays out a bold vision of a privacy preserving architecture for personal data sharing in the cloud based on trusted intermediaries. The second paper by Chen & Sion evaluates the economic viability of implementing secure outsourced data management in untrusted clouds. Their thesis that today's cryptography and security solutions are simply not expressive enough to support outsourcing in a cost-effective way can be viewed as a call for innovative approaches that indeed offer practical solutions.

The next three papers focus specifically on cloud as a vehicle to offloading the complex task of information sharing. Nabeel & Bertino address an important problem of fine-grained access control based on a broadcast based group key management scheme that provides a scalable solution to selectively share data with others in an untrusted cloud. Anh and Datta define a design space for fine-grained access control solutions based on the level of access, the trust one has in the cloud, and how the work is split between the client and the cloud - this provides an elegant approach to viewing current solutions and exploring future challenges. Hamlen, Kagal, and Kantarcioglu identify an approach to policy enforcement wherein application code self-censor their resource accesses to implement efficient access control.

The following three papers explore mechanisms for secure data processing in the cloud. Khadilkar, Oktay, Kantarcioglu, and Mehrotra explores the design space for data and workload partitioning in hybrid clouds wherein in-house computing resources are integrated with public cloud services to support a secure and economical data processing solution. Mukundan, Madria, and Linderman tackle the challenge of data integrity, specifically, provable data possession in the presence of multiple replicas that enables owners to verify that cloud providers maintain the requisite number of replicas for data availability based on the service level agreement. Arasu, Blanas, Eguro, et. al. describe the Cipherbase relational database technology they are building at Microsoft that leverages novel customized hardware to store and process encrypted data. The last three papers in the series explore cloud in the role of applications as a service. Feldman, Blankstein, Freedman, and Felton introduce two cloud deployable application frameworks – SPORC (for collaborative applications such as text editor and shared calendars) and Frientegrity (that extends SPORC to online social networking) that do not require users to trust cloud providers with either confidentiality or integrity of data. Sanamrad, Wider, et. al. introduce a middleware approach that enables users to encrypt and store calendar entries and emails in Google Calendar and Gmail. Finally, the paper by De Cristofaro, Soriente, Tsudik and Williams describes a system they call hummingbird that implements a functionality similar to twitter using which users can tweet, follow and search while simultaneously protecting the tweet content, hashtags, and follower content from being exposed to the hummingbird service provider.

As we have become accustomed to in reading the Data Engineering Bulletins, the range of articles differ significantly in the level of their depth and treatment of the subject - while some lay out a vision for the future, other offer technically mature approaches based on significant prior work by the authors. Irrespective of the nature of the papers, collectively they provide a good summary of the state-of-the-art research and also a wealth of interesting new ideas/thoughts. This bulletin makes a wonderful reading for anyone interested in either learning about or intending to do research on what is possibly one of the most important challenges for computer science in the next decade.

Finally, I would like to acknowledge the generous help by Kerim Yasin Oktay in meticulously following up with the authors, collecting, formatting, and compiling the papers into the bulletin.

Sharad Mehrotra
University of California, Irvine

On the Trusted Use of Large-Scale Personal Data

Yves-Alexandre de Montjoye* #1, Samuel S. Wang *2, Alex (Sandy) Pentland #3
#The Media Laboratory
*Decentralized Information Group, CSAIL
Massachusetts Institute of Technology
Cambridge, MA, USA
1yva@mit.edu, 2samuelsw@csail.mit.edu, 3pentland@mit.edu

Abstract

Large-scale personal data has become the new oil of the Internet. However, personal data tend to be monopolized and siloed by online services which not only impedes beneficial uses but also prevents users from managing the risks associated with their data. While there is substantial legal and social policy scholarship concerning ownership and fair use of personal data, a pragmatic technical solution that allows governments and companies easy access to such data and yet protects individual rights has yet to be realized and tested. We introduce openPDS, an implementation of a Personal Data Store, which follows the recommendations of the WEF, the US NSTIC and the US Consumer Privacy Bill of Rights. openPDS allows users to collect, store, and give fine-grained access to their data in the cloud. openPDS also protects users' privacy by only sharing anonymous answers, not raw data. Indeed, a mechanism to install third-parties applications on a user's PDS allows for the sensitive part of the data processing to take place within the PDS. openPDS can also engage in privacy-preserving group computations to aggregate data across users without the need to share sensitive data with an intermediate entity.

1 Motivation

Personal Data has become the new oil of the Internet [1], and the current excitement about Big Data is increasingly about the analysis of personal data: location data, purchasing data, telephone call patterns, email patterns, and the social graphs of LinkedIn, Facebook, and Yammer. However, currently personal data is mostly siloed within large companies. This prevents its use by innovative services and even by the user who generated the data. The problem is that while there is substantial legal and social policy scholarship concerning ownership and fair use of personal data, a pragmatic technical solution that allows governments and companies easy access to such data and yet protects individual rights and privacy has yet to be realized and tested.

We thus an architecture for the trusted use of large-scale personal data that is consistent with new "best practice" standards which require that individuals retain the legal rights of possession, use, and disposal for data that is about them. To do this, we develop openPDS–an open-source Personal Data Store enabling the user to collect, store, and give access to their data while protecting their privacy. Via an innovative framework

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

for installing third-party applications, the system ensures that most processing of sensitive personal data takes place within the user's PDS, as opposed to a third-party server. The framework also allows for PDSs to engage in privacy-preserving group computation, which can be used as a replacement for centralized aggregation.

Although our aim is to provide a technical solution, it is important for such solution to be not only compatible but also aligned with political and legal thinking. openPDS is compatible with and incorporates best practice suggestions of the US Consumer Privacy Bill of Rights [2], the US National Strategy for Trust Identities in Cyberspace (NSTIC) [3], the Department of Commerce Green Paper, and the Office of the Presidents International Strategy for Cyberspace [4]. In addition, it follows the Fair Information Practices (FIPs) which have mandated that personal data be made available to individuals upon request. In addition openPDS is aligned with the European Commission's 2012 reform of the data protection rules [5]. This reform redefines personal data as "any information relating to an individual, whether it relates to his or her private, professional or public life." It also states the right for people to "have easier access to their own data and be able to transfer personal data from one service provider to another more easily" as well as a right to be forgotten. All these ideas and regulations recognize that personal data needs to be under the control of the user in order to avoid a retreat into secrecy where these data become the exclusive domain of private companies, denying control to the user.

2 Personal Data Stores (PDS)

Many of the initial and critical steps towards implementation of these data ownership policies are technological. The user needs to have control of a secured digital space, a personal data store (PDS), where his data can live. Given the huge number of sources of data that a user interacts with every day, mere interoperability is not enough. There needs to be a centralized location that a user is able to view and reason about the data that is collected about himself. The PDS should allow the user to easily control the flow of data and manage fined grained authorizations for third-service services, fulfilling the vision of the New Deal on Data [1]. A PDS-based market is likely to be fair, as defined by the Fair Information Principles, as the user is the one controlling the access to his data. The user can decide whether such services provide enough value compared to the amount of data it asks for; the user can ask questions like "Is finding out the name of this song worth enough to me to give away my location?" The PDS will help the user make the best decision for himself. Using a privacy-preserving PDS allows for greater data portability, as the user can seamlessly interface new services with his PDS, and will not lose ownership or control of his personal data.

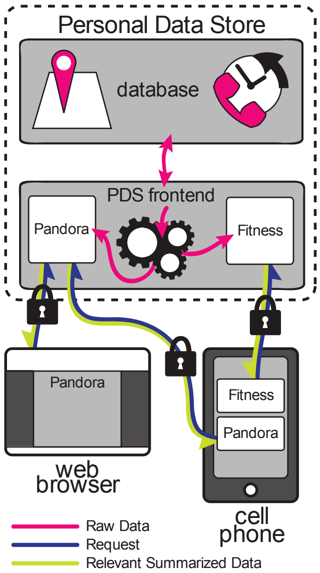
Thanks to the policy requirement of data portability, a PDS-based data market is likely to be economically efficient, as the system removes barriers to entry for new businesses. It allows the more innovative companies to provide better data-powered services. The services chosen by the user will have access to historical data, which was potentially collected even before the creation of the service. Moreover, the services will not be forced to collect data themselves, as they will have access to data coming from other apps. Service providers can thus concentrate on delivering the best possible experience to the user. For example, a music service could provide you a personalized radio station, leveraging the songs and artists you said you like across the web, what your friends like, or even which nightclubs you go to. The real value of large-scale data appears when innovators can create data driven applications on top of rich personal user information.

3 Question Answering Framework

In the existing mobile space, personal data is offloaded from mobile devices onto servers owned by the application creator. This model prevents users from being able to control their own data; once they hand that data over to a corporation, it is difficult or impossible to refute or retract.

The key innovation in the openPDS model is that computations on user data are performed in the safe environment of the PDS, under the control of the user. The idea is that only the relevant summarized data for providing functionality to the application should leave the boundaries of the user's PDS [See Fig. 1].

Figure 1: openPDS system's architecture.



Rather than exporting raw GPS coordinates, it could be sufficient for an app to know which general geographic zone you are currently in. Instead of sending raw GPS coordinates to the app owner's server to process, that computation can be done by the PDS app in the user's PDS server. The system is still exposing personal data of the user, but it is constrained to be what the app strictly needs to know, rather than the raw data objects the user generates. A series of such computed answer would also be easier to anonymized than high-dimensional sensor data. App designers would take care to declare to users as well as in a machine readable format to be enforced exactly what data is being computed over, what inferences are being exposed to external apps, and what data is being reported back to the company's servers.

With this model of computation, it is relatively easy to monitor the communication between a PDS app and its Android counterpart. Since the user owns the platform on which the PDS app executes, it is possible to eavesdrop on the data that is exposed by the PDS app to the Android app. If an app is accessing and exporting more data than it is supposed to be in order to provide the required services, it will be known by people who use the app, and could potentially be reflected in the app's reviews. This ability to monitor the results of computation on user data provides a coarse way to verify that one's personal data is not being unexpectedly leaked.

4 The user experience

If Alice chooses to download a PDS-aware version of Spotify, the music streaming service, she would install it just like she would any other Android application. Upon launching the application, the Android app would prompt her to install a Spotify app onto her PDS. The description of the PDS app would describe exactly what data Spotify would access and reason over on her PDS, as well as what relevant summarized information is passed on to Spotify's servers, for example to offer personalized music radios to the user. This allows Alice to understand what it means for her privacy to install the app.

When using the Spotify Android app, rather than storing Alice's personal data on Spotify's servers, the Spotify PDS app would instead access and process the data on Alice's PDS. Alice would have installed a PDS instance on her favorite cloud provider, or on her own server. Over time, her PDS would be filled with information collected by her phone, but also information about her musical tastes, her contacts, as well as a stream of other sensor information that Alice accumulates in her day to day life. Alice would have full control over this data, and could see exactly what data her phone, other sensors, and services gathers about her over time.

Because the Spotify PDS app is being run on a computing infrastructure that Alice owns, the outgoing data can be audited to verify that no unexpected data is escaping the boundaries of her PDS. In this way, rich applications and services can be built on top of the PDS that leverage all of these disparate data sources, while Alice still owns the underlying data behind these computations, and can take steps to preserve aspects of her privacy.

5 Key Research Questions

This vision is a world in which personal data that is easily available but yet the individual is protected. There are many technical challenges to accomplish this vision. For instance, the question-and-answer mechanism that allows certified answers to be shared instead of raw data requires the development of new privacy preserving technologies for user-centric on-the-fly anonymization.

Similarly, auditing the distribution and sharing of information in order to confirm that all data sharing is as intended requires the development of new algorithms and techniques to detect breaches and attacks.

There are also significant user interface questions, so that users really understand the risks and rewards they will be asked to opt into and are not overwhelmed with choices. A key idea for these interface questions is to use experimentation to determine user preferences for risk/reward, assessed via mechanisms such as differential privacy, in this question-answering environment.

6 Conclusion

As technologists and scientists, we are convinced that there is amazing potential in personal data, but also that the user has to be in control, making the trade-off between risks and benefits of data uses. openPDS is one attempt to provide a privacy-preserving Personal Data Store that makes it easy and safe for the user to own, manage and control his data. By anonymously just answering questions on-the-fly, openPDS opens up a new way for individuals to regain control over their data and privacy while supporting the creation of smart, data-driven applications.

References

[1] Personal Data: The Emergence of a New Asset Class, `http://www3.weforum.org/docs/WEF_ITTC_PersonalDataNewAsset_Report_2011.pdf`.

[2] US Consumer Privacy Bill of Rights, `http://www.whitehouse.gov/sites/default/files/privacy-final.pdf`

[3] Reality Mining of Mobile Communications: Toward a New Deal on Data. `https://members.weforum.org/pdf/gitr/2009/gitr09fullreport.pdf`

[4] National Strategy for Trust Identities in Cyberspace, `http://www.whitehouse.gov/sites/default/files/rss_viewer/NSTICstrategy_041511.pdf`

[5] International Strategy for Cyberspace, `http://www.whitehouse.gov/sites/default/files/rss_viewer/internationalstrategy_cyberspace.pdf`

[6] European Commission proposes a comprehensive reform of data protection rules to increase users' control of their data and to cut costs for businesses, `http://europa.eu/rapid/pressReleasesAction.do?reference=IP/12/46&format=HTML&aged=0&language=EN&guiLanguage=en`

On Securing Untrusted Clouds with Cryptography

Yao Chen, Radu Sion
Stony Brook Network Security and Applied Cryptography Lab
{yaochen,sion}@cs.stonybrook.edu

Abstract

In a recent interview, Whitfield Diffie argued that "the whole point of cloud computing is economy" and while it is possible in principle for "computation to be done on encrypted data, [...] current techniques would more than undo the economy gained by the outsourcing and show little sign of becoming practical*". Here we explore whether this is truly the case and quantify just* how *expensive it is to secure computing in untrusted, potentially curious clouds.*

We start by looking at the economics of computing in general and clouds in particular. Specifically, we derive the end-to-end cost of a CPU cycle in various environments and show that its cost lies between 0.5 picocents in efficient clouds and nearly 27 picocents for small enterprises (1 picocent = \$1 $\times$ 10^{-14}), values validated against current cloud pricing.

We then explore the cost of common cryptography primitives as well as the viability of their deployment for cloud security purposes. We conclude that Diffie was correct. Securing outsourced data and computation against untrusted clouds is indeed costlier than the associated savings, with outsourcing mechanisms up to several orders of magnitudes costlier than their non-outsourced locally run alternatives.

1 Introduction

Commoditized outsourced computing has finally arrived, mainly due to the emergence of fast and cheap networking and efficient large scale computing. Amazon, Google, Microsoft and Oracle are just a few of the providers starting to offer increasingly complex storage and computation outsourcing. CPU cycles have become consumer merchandise.

In [10] we explored the end-to-end cost of a CPU cycle in various environments and show that its cost lies between 0.45 picocents in efficient clouds and 27 picocents for small business deployment scenarios (1 picocent = \$1 $\times$ 10^{-14}). In terms of pure CPU cycle costs, current clouds present seemingly cost-effective propositions for personal and small enterprise clients.

Nevertheless, cloud clients are concerned with the **privacy of their data and computation** – this is often the primary adoption obstacle, especially for medium and large corporations, who often fall under strict regulatory compliance requirements. To address this, existing secure outsourcing research addressed several issues including guaranteeing integrity, confidentiality and privacy of outsourced data to secure querying on outsourced encrypted database. Such assurances will likely require strong cryptography as part of elaborate intra- and client-cloud protocols. Yet, strong crypto is expensive. Thus, it is important to ask: how much cryptography can we afford in the cloud while maintaining the cost benefits of outsourcing?

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Some believe the answer is simply *none*. For example, in a recent interview [48] Whitfield Diffie argued that "**current techniques would more than undo the economy [of] outsourcing and show little sign of becoming practical**."

Here we set out to find out whether this holds and if so, by what margins. One way to look at this is in terms of CPU cycles. For each desired un-secured client CPU cycle, *how many additional cloud cycles can we spend on cryptography*, before its outsourcing becomes too expensive? We end up gaining the insight that today's secure data outsourcing primitives are often orders of magnitude more expensive than local execution, mainly due to the fact that we do not know how to process complex functions on encrypted data efficiently enough. And outsourcing simple operations – such as existing research in querying encrypted data, keyword searches, selections, projections, and simple aggregates – is simply not profitable. Thus, while traditional security mechanisms allow the elegant handling of inter-client and outside adversaries, today it is still too costly to secure against cloud insiders with cryptography.

2 Cost Models

Parameters	H	S	M	L
CPU utilization	5-8%	10-12%	15-20%	40-56%
server:admin ratio	N.A.	100-140	140-200	800-1000
Space (sqft/month)	N.A.	$0.5	$0.5	$0.25
PUE	N.A.	2-2.5	1.6-2	1.2-1.5

Figure 1: Sample key parameters.

To reach the granularity of computing cycles, in [10] we explore the cost of running computing at different levels. We chose environments of increasing size: home, small enterprises, mid-size and large size data centers. The boundaries between these setups are often dynamic and the main reason we're using them is to help differentiate a set of key parameters (Figure 1).

2.1 Levels

Home Users (H). We include this scenario as a baseline for a simple home setup containing several computers. This could correspond to individuals with spare time to maintain a small set of computers, or a small home-based enterprise without staffing costs.

Small Enterprises (S). We consider here any scenario involving an infrastructure of up to 1000 servers run in-house in a commercial enterprise. The cost structure will start to feature most of the usual suspects, including commercial energy and network pricing, cooling, space leases, staffing etc. Small enterprises can not afford custom hardware, efficient power-distribution, and cooling or dedicated buildings among others. More importantly, in addition to power distribution inefficiencies, due to their nature, small enterprises cannot be run at high utilization as they would be usually under the incidence of business cycles and its associated peak loads.

Mid-size Enterprises (M). We consider here setups of up to 10,000 servers, run by a corporation, often in its own dedicated data center(s). Mid-size enterprises might have some clout and access to better service deals for network service as well as more efficient cooling and power distribution. They are not fully global, yet could feature several centers across one or two time zones, allowing increased independence from local load cycles as well as the ability to handle daily peaks better by shifting loads across timezones. All the above results ultimately in increased utilization (20-25% est.) and overall efficiency.

Large Enterprises/Clouds (L). Clouds and large enterprises run over 10,000 servers, cross multiple timezones, often literally at a global level, with large data centers distributed across all continents and often in tens to hundreds of countries. For example Google has built a 30-acre site in Dalles, Oregon, next to a hydroelectric dam providing cheap power. The site is composed of 34,000 square feet buildings [26]. Especially in cloud setups, high speed networks allow global-wide distribution and integration of load from thousands of individual points of load. This in turn flattens the 24-hour overall load curve and allows for efficient peak handling and comparably high utilization factors (50-60% est. [22]). Cloud providers run the most efficient infrastructures, and often are at the forefront of innovation. In one notorious instance, Google for example asked Intel for chips tolerating more heat, to allow for a few degrees increase in data center operating temperatures - which in turn increases cooling efficiency by whole percentage points [36]. Moreover, clouds have access to bulk-pricing for network service from large ISPs, often one order of magnitude cheaper than mid-size enterprises.

2.2 Factors

We now consider the cost factors that come into play across all of the above levels. These can be divided into a set of inter-dependent vectors, including: hardware (servers, networking gear), building (floor space leasing), energy (running hardware and cooling), service (administration, staffing, software maintenance), and network service. Other breakdown layouts of these factors are possible.

Server Hardware. Hardware costs include servers, racks, power equipment, network equipment, cooling equipment etc. We will discuss network equipment later.

We note that these costs drop with time, likely even by the time this goes to print. For example, while many of the current documented mid-size deployments use single or multi-CPU System-X blade servers at around $1-2000 each [25], large data centers deploy custom setups at about $3000 for 4 CPUs, near-future developments could yield important changes. [1] We will be conservative and empirically assume home PC prices of around $750/CPU, small and mid-size enterprise costs of around $1000/CPU (for 2 CPU blades) and cloud-level costs of no more than $500/CPU.

Energy. Energy in data centers does not only include power, computing and networking hardware but the entire support infrastructure, including cooling, physical security, and overall facilities. A simple rough way to infer power costs is by estimating the Power Usage Efficiency (PUE) of the data center. The PUE is a metric defined by the GreenGrid Consortium to evaluate the energy efficiency of a data center [20] (PUE = Total Power Usage / IT Equipment Power Usage).

We will assume 1.2-1.5 PUE for large enterprises, 1.6-2 PUE for mid-size enterprises and 2-2.5 for small enterprises [38]. Costs of electricity are relatively uniform and documented [2].

Service. Evaluating the staffing requirements for data centers is an extremely complex endeavor as it involves a number of components such as software development and management, hardware repair, maintenance of cooling, building, network and power services.

Analytical approaches are challenged by the sparsity of available relevant supporting data sets.

We deployed a set of commonly accepted rule of thumb values that have been empirically developed and validate well [23]: the server to administrator ratio varies from 2:1 up to experimental 2500:1 values due to different degrees of automation and data management. In deployment, small to mid-size data centers feature a ratio of 100-140:1 whereas cloud level centers can go up to 1000:1 [19, 22].

Network Hardware. To allow for analysis of network intensive protocols, we chose to separate network transport service costs from the other factors of impact in the bottom line for CPU cycle. Specifically, while the internal network infrastructure costs will be factored in the data center costs, network service will not. We will estimate separately the cost of transferring a bit reliably to/from the data center intermediated by outside ISPs' networks. Internal network infrastructure costs can be estimated by evaluating the number of required switches and routers. The design of scalable large economy network topology with high inter-node bandwidth for data centers is an ever ongoing research problem [39]. We base our results on some of the latest state of the art research, deploying fat tree interconnect structures. Fat trees have been shown to offer significantly lower overall hardware costs with good overall connectivity factors.

Floor Space. Floor space costs vary wildly, by location and use. While small to mid-size enterprises usually have data centers near their location (thus sometimes incurring office-level pricing), large companies such as Google and Microsoft tend to build data centers on owned land, in less populated place where the per sqft price can be brought down much lower, often amortized to zero over time.

[1] In one documented instance, e.g., Amazon is working with Rackable Systems to deliver an under $700 AMD-based 6 CPU board dubbed CEMS (Cooperative Expendable Micro-Slice Servers) V3.

$$
\begin{aligned}
CycleCost &= \frac{Server + Energy + Service + Network + Floor}{Total\ Cycles} \\
&= \frac{\lambda_s \cdot N_s/\tau_s + (w_p \cdot \mu + w_i \cdot (1-\mu)) \cdot PUE \cdot \lambda_e + \frac{N_s}{\alpha} \cdot \lambda_p + \lambda_w \cdot N_w/\tau_w + \lambda_f \cdot \frac{(w_p \cdot \mu + w_i \cdot (1-\mu)) \cdot PUE}{\beta}}{\mu \cdot \nu \cdot N_s}
\end{aligned} \tag{1}
$$

We also note that floor surface is directly related to power consumption and cooling with designs supporting anywhere from 40 to 250 watt/sqft [17]. Thus, the overall power requirements (driven by CPUs) impact directly the required space.

2.3 The Costs

We start by evaluating the amortized dollar cost of a CPU cycle in equation (1). See notations in Figure 2 and various setups' parameters in Figure 1.

Symbol	Definition
N_s, N_w	number of servers,switches
α	administrator : server ratio
β	watt per sq ft
λ_s, λ_w	server,switch price
λ_p, λ_f	personnel,floor cost/sec
λ_e	electricity price/(watt·sec)
μ	CPU utilization
ν	CPU frequency
τ_s, τ_w	servers,switches lifespan (5 y.)
w_p, w_i	server power at peak,idle

Figure 2: Notations for (1).

Provider	**Picocents**
Amazon EC2	0.93 - 2.36
Google AppEngine	up to 2.31
Microsoft Azure	up to 1.96

Figure 3: Current pricings.

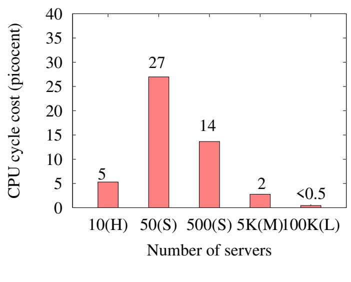


Figure 4: CPU cycle costs

"CPU cycles" are architecture-specific, yet we chose them to evade higher level, semantic-dependent units such as application-specific 'transactions". When reasoning about general computing it is not clear what types of higher-level transactions are appropriate to consider as 'units". After all, the dollar cost of serving an HTTP 'transaction" is only marginally relevant in evaluating the end-to-end costs of cloud-hosting arbitrary applications, across different infrastructures and languages. And, we will show that CPU cycles validate well as a consistent unit – probably in no small part due to the recent (almost) universality of x86 platforms across all environments, in effect reducing the impact of architecture specificity.

The results are depicted in Figure 4, costs ranging from 0.45 picocents/cycle in very large cloud settings all the way to (S), the costliest environment, where a cycle costs up to 27 picocents (*1 US picocent* = *$1 $\times 10^{-14}$*). We validate our results by exploring the pricing of the main cloud providers (Figure 3). The prices lie surprisingly close to each other and to our estimates, ranging from 0.93 to 2.36 picocents/cycle. The difference in cost is due to the fact that these points include not only CPUs but also intra-cloud networking, instance-specific disk storage and cloud providers' profit.

Storage Cost. Simply storing bits on disks has become truly cheap. Increased hardware reliability (with mean time between failures rated routinely above a million hours even for consumer markets) and economies of scale resulted in extreme drops in the costs of disks. In [10], we showed that in terms of amortized acquisition costs, the best price/hardware/MTBF ratio from our sample set is at 26.06 picocents/bit/year. The dominant factor is energy, 60-350 picocents/bit/year, at 60-90% of the total cost. The lowest total cost from our sample set is at about 100 picocents/bit/year.

Network Service Published network service cost numbers place network service costs for large data centers at around $13/ Mbps/ mo and for mid-size setups at $95/Mbps/mo [22] for *guaranteed* bandwidth. Home user and small enterprise pricing benefits from economies of scale, e.g., Optimum Online provides 15/5 Mbps internet connection for small business starting at $44.9/ mo [43]. Yet we note that the quoted bandwidth is not guaranteedand refers only to the hop connecting the client to the provider. Figure 5 summarizes network service cost in the four environments. When inferring the per-bit transmission costs we considered the uplink/downlink costs were independently priced at the same total price quoted for the entire connection.

	AES-128	AES-192	AES-256
S	1.42E+03	1.48E+03	1.52E+03
L	2.37E+01	2.47E+01	2.53E+01

Figure 6: AES-128, AES-192, AES-256 costs (per byte) on 64-byte input.

	1024 bit		2048 bit	
	Encrypt	Decrypt	Encrypt	Decrypt
S	3.74E+06	1.03E+08	8.99E+06	6.44E+08
L	6.24E+04	1.72E+06	1.50E+05	1.07E+07

Figure 7: Cost of RSA encryption/decryption on 59-byte messages. (picocents)

	1024 bit		2048 bit	
	Sign	Verify	Sign	Verify
S	5.73E+07	6.94E+07	1.89E+08	2.30E+08
L	9.55E+05	1.16E+06	3.15E+06	3.84E+06

Figure 8: DSA on 59-byte messages. The 1024-bit DSA uses 148-byte secret key and 128-byte public key. The 2048-bit DSA uses 276-byte secret key and 256-byte public key.

In other words, we assumed the provider would charge the same amount for only the uplink connection.

	H, S	M	L
monthly	$44.90	$95	$13
bandwidth (d/u)	15/5 Mbps	per 1Mbps	per 1Mbps
dedicated	No	Yes	Yes
picocent/bit	115/345	3665	500

Figure 5: Summarized network service costs [10].

The end-to-end cost of network transfer includes the cost on both communicating parties and the CPU overheads of transferring a bit from one application layer to another. Moreover, for reliable networking (e.g., TCP/IP) we need to also factor in the additional traffic and spent CPU cycles (e.g., SYN, SYN/ACK, ACK, for connection establishment, ACKs for sent data, window management, routing, packet parsing, re-transmissions). In the $S \to L$ scenario, it costs more than 900 picocents to transfer one bit reliably.

3 Cryptography

So far we know that a CPU cycle will set us back 0.45-27 picocents, transferring a bit costs at least 900 picocents, and storing it costs under 100 picocents/year. We now explore the costs of basic crypto and modular arithmetic. All values are in picocents. Note that CPU cycles needed in cryptographic operations often vary with optimization algorithms and types of hardware used (e.g., specialized secure CPUs and crypto accelerators with hardware RSA engines [1] are cheaper per cycle than general-purpose CPUs).

Symmetric Key Crypto. We first evaluate the per-bit costs of AES-128, AES-192, AES-256 and illustrate in Figure 6. The evaluation is based on results from the ECRYPT Benchmarking of Cryptographic Systems (eBACS) [6].

RSA. Using modular exponentiation, RSA public key encryption takes $O(k^2)$, private key decryption $O(k^3)$, and key generation $O(k^4)$ steps, where k is the number of bits in the modulus [28]. Numerous algorithms aim to improve the speed of RSA, mainly by reducing the time to do modular multiplications. In Figure 7, we illustrate the costs of RSA encryption/decryption using benchmark results from [6].

PK Signatures. We illustrate costs of DSA, and ECDSA signatures based on NIST elliptic curves [6] in Figures 8, 9.

Cryptographic Hashes We also show per byte cost of MD5 and SHA1 with varied input sizes.

4 Secure Outsourcing

Thus armed with an understanding of computation, storage, network and crypto costs, we now ask whether securing cloud computing against insiders is a viable endeavor.

	ECDSA-163		ECDSA-409	
	KG/SGN	Verify	KG/SGN	Verify
S	1.36E+08	2.65E+08	9.60E+08	1.91E+09
L	2.27E+06	4.41E+06	1.60E+07	3.19E+07
	ECDSA-571			
	KG/SGN		Verify	
S	2.09E+09		4.18E+09	
L	3.48E+07		6.96E+07	

Figure 9: Costs of ECDSA signatures on 59-byte messages (curve over a field of size 2^{163}, 2^{409}, 2^{571} respectively). (picocents)

	MD5		SHA1	
	4096	64	4096	64
S	1.52E+02	3.75E+02	2.14E+02	6.44E+02
L	2.53E+00	6.25E+00	3.56E+00	1.07E+01

Figure 10: Per-byte cost of MD5 and SHA1 (with 64-byte and 4096-byte input).

We start by exploring what security means in this context. Naturally, the traditional usual suspects need to be handled in any outsourcing environment: (mutual) authentication, logic certification, inter-client isolation , network security as well as general physical security. Yet, all of these issues are addressed extensively in existing infrastructures and are not the subject of this work.

Similarly, for conciseness, within this scope, we will isolate the analysis from the additional costs of software patching, peak provisioning for reliability, network defenses etc.

4.1 Trust

We are concerned cloud clients being often reluctant to place sensitive data and logic onto remote servers without guarantees of compliance to their security policies [15, 29]. This is especially important in view of recent sub-poenas and other security incidents involving cloud-hosted data [11, 12, 35]. The viability of the cloud computing paradigm thus hinges directly on the issue of clients' trust and of major concern are cloud insiders. Yet how "trusted" are today's clouds from this perspective? We identify a set of scenarios.

Trusted clouds. In a *trusted* cloud, in the absence of unpredictable failures, clients are served correctly, in accordance to an agreed upon service contract and the cloud provider's policies. No insiders act maliciously.

Untrusted clouds. For *untrusted* clouds, we distinguish several cases depending on the types of illicit incentives existing for the cloud and the client policies with which these will directly conflict. We call a cloud *data-curious* if insiders thereof have incentives to violate confidentiality policies (mainly) for (sensitive) client data. Similarly, in an *access-curious* cloud, insiders will aim to infer client access patterns to data or reverse-engineer and understand outsourced computation logic. A *malicious* cloud will focus mainly on (data and computation) integrity policies and alter data or perform incorrect computation.

Reasonable cloud insiders are likely to factor in the potential illicit gains (the incentives to violate the policy), the penalty for getting caught, as well as the probability of detection. Thus for most practical scenarios, insiders will engage in such behavior only if they can get away undetected with high probability, e.g., when no (cryptographic?) safeguards are in place to enable the detection.

4.2 Secure Outsourcing

Yet, millions of users embrace free web apps in **an untrusted provider model**. This shows that today's (mostly personal) cloud clients are willing to trade their privacy for (free) service. This is not necessarily a bad thing, especially at this critical-mass building stage, yet raises questions of clouds' viability for commercial, regulatory-compliant deployment, involving sensitive data and logic. And, from a bottom-line cost-perspective, is it worth even trying? This is what we aim to understand here.

In the following **we will assess whether clouds are economically tenable if their users do not trust them and therefore must employ cryptography and other mechanisms to protect their data.** A number of experimental systems and research efforts address the problem of outsourcing *data* to *untrusted service providers*, including issues ranging from searching in remote encrypted data to guaranteeing integrity and confidentiality to querying of outsourced data. In favor of cloud computing, we will set our analysis in the most favorable $S \rightarrow L$ scenario, which yields most CPU cycle savings.

4.2.1 The Case for Basic Outsourcing

Before we tackle cloud security, let us look at the simplest computation outsourcing scenario (where clients outsource data to the cloud, expect the cloud to process it, and send the results back). In existing work [10], we show that, to make (basic, unsecured) outsourcing cost effective, the cost savings (mainly from cheaper CPU cycles) need to outweigh the cloud's distance from clients. In $S \rightarrow L$, outsourced tasks should perform at least 1,000 CPU cycles per every 32 bit data, otherwise it is not worth outsourcing them.

4.2.2 Encrypted Data Storage with Integrity

With an understanding of the basic boundary condition defining the viability of outsourcing we now turn our attention to one of the most basic outsourcing scenarios in which a single data client places data remotely for simple storage purposes. In the $S \rightarrow L$ scenario, the amortized cost of storing a bit reliably either locally *or remotely* is under 9 picocents/month (including power). Network transfer however, is of at least 900 picocents per accessed bit, a cost that is not amortized and two orders of magnitude higher.

From a technological cost-centric point of view it is simply not effective to store data remotely: **outsourced storage costs can be upwards of 2+ orders of magnitude higher than local storage** for the $S \rightarrow L$ scenario *even in the absence of security assurances*.

Cost of Security. Yet, outsourced storage providers exist and thrive. This is likely due to factors outside of our scope, such as the convenience of being able to have access to the data from everywhere or collaborative application scenarios in which multiple data users share single data stores (multi-client settings). Notwithstanding the reason, since consumers have decided it is worth paying for outsourced storage, the next question we ask is, how much more would security cost in this context? We first survey some of the existing work.

Several existing systems encrypt data before storing it on potentially data-curious servers [7, 9, 37]. File systems such as I^{3}FS [27], GFS [18], and Checksummed NCryptfs [46] perform online real-time integrity verification.

It can be seen that two main assurances are of concern here: integrity and confidentiality. The cheapest integrity constructs deployed in most of the above revolve around the use of hash-based MACs. As discussed above, SHA-1 based keyed MAC constructs with 4096-byte blocks would cost around 4 picocent/byte on the server and 200 picocents/byte on the client side, leading to a total cost of about 25 picocents/bit. This is at least 4 times lower than the cost of storing the bit for a year and at least one order of magnitude lower than the costs incurred by transferring the same bit (at 900+ picocents/bit). Thus, **for outsourced storage, integrity assurance overheads are negligible.**

For publicly verifiable constructs, crypto-hash chains can help amortize their costs over multiple blocks. In the extreme case, a single signature could authenticate an entire file system, at the expense of increased I/O overheads for verification. Usually, a chain only includes a set of blocks.

For an average of twenty 4096 byte blocks[2] secured by a single hash-chain signed using 1024-bit RSA, would yield an amortized cost approximately 1M picocents per 4096-byte block (30+ picocents/bit) for client read verification and 180+ picocents/bit for write/signatures. This is up to **8 times more expensive than the MAC based case**.

[2]Douceur et al. [16], show that file sizes can be modeled using a log-normal distribution. E.g., for $\mu^e = 8.46$, $\sigma^e = 2.4$ and 20,000 files, the median file size would be 4KB, mean 80KB, along with a small number of files with sizes exceeding 1GB [3, 16].

4.2.3 Searches on Encrypted Data

Confidentiality alone can be achieved by encrypting the outsourced content before outsourcing to potentially access-curious servers. Once encrypted however, it cannot be easily processed by servers.

One of the first processing primitives that has been explored allows clients to search directly in remote encrypted data [4, 5, 13]. In these efforts, clients either linearly process the data using symmetric key encryption mechanisms, or, more often, outsource additional secure (meta)data mostly of size linear in the order of the original data set. This meta-data aids the server in searching through the encrypted data set while revealing as little as possible.

But is remote searching worth it vs. local storage? We concluded above that simply using a cloud as a remote file server is extremely non-profitable, up to several orders of magnitude. Could the searching application possibly make a difference? This would hold if either (i) the task of searching would be extremely CPU intensive allowing the cloud savings to kick in and offset the large losses due to network transfer, or (ii) the search is extremely selective and its results are a very small subset of the outsourced data set – thus amortizing the initial transfer cost over multiple searches.

We note that existing work does not support any complex search predicates outside of simple keyword matching search. Thus the only hope there is that the search-related CPU load (e.g., string comparison) will be enough cheaper in the cloud to offset the initial and result transfer costs.

Keyword searching can be done in asymptotically constant time, given enough storage or logarithmic if B-trees are used. While the client could maintain indexes and only deploy the cloud as a file server, we already discovered that this is not going to be profitable. Thus if we are to have any chance to benefit here, the index structures need to also be stored on the server.

In this case, the search cost includes the CPU cycle costs in reading the B-tree and performing binary searches within B-tree nodes. As an example, consider 32 bit search keys (e.g., as they can be read in one cycle from RAM), and a 1 TB database. 1-3 CPU cycles are needed to initiate the disk DMA per reading, and each comparison in the binary search requires another 1-3 cycles (for executing a comparison conditional jump operation). A B-tree with 16KB nodes will have approximately a 1000 fanout and a height of 4-5, so performing a search on this B-tree index requires about 100-300 CPU cycles. Thus in this simple remote search, $S \to L$ outsourcing would result in CPU-related savings of around 2,500-8,000 picocents per access. Transferring 32 bits from $S \to L$ costs upwards of 900 picocents. Outsourced searching becomes thus more expensive for any results upwards of 36 bytes per query.

4.2.4 Insights into Secure Query Processing

By now we start to suspect that similar insights hold also for outsourced query processing. This is because we now know that (i) the tasks to be outsourced should be CPU-intensive enough to offset the network overhead – in other words, outsourcing peanut counting will never be profitable, and (ii) existing confidentiality (e.g., homomorphisms) and integrity (e.g., hash trees, aggregated signatures, hash chains) mechanisms can "secure" only very simple basic arithmetic (addition, multiplication) or data retrieval (selection, projection) which would cost under a few of cycles per word if done in an unsecured manner. In other words, *we do not know yet how to secure anything more complex than peanut counting*. And outsourcing of peanut counting is counter productive in the first place. Ergo our suspicion.

We start by surveying existing mechanisms. Hacigumus et al. [21] propose a method to execute SQL queries over partly obfuscated outsourced data to protect data **confidentiality** against a data-curious server. The main functionality relies on (i) partly obfuscating the outsourced data by dividing it into a set of partitions, (ii) query rewriting of original queries into querying referencing partitions instead of individual tuples, and (iii) client-side pruning of (necessarily coarse grained) results. The information leaked to the server is balancing a trade-off between client-side and server-side processing, as a function of the data segment size. [24] explores optimal bucket sizes for certain range queries.

Ge et al. [47] discuss executing aggregation queries with confidentiality on an untrusted server. Unfortunately, due to the use of extremely expensive homomorphisms this scheme leads to large processing times for any reasonably security parameter settings (e.g., for 1024 bit fields, 12+ days *per query* are required).

Other researchers have explored the issue of **correctness** in settings with potentially malicious servers. In a publisher-subscriber model, Devanbu et al. deployed Merkle trees to authenticate data published at a third party's site [14], and then explored a general model for authenticating data structures [32, 33]. In [40, 41] as well as in [31], mechanisms for efficient integrity and origin authentication for selection predicate query results are introduced. Different signature schemes (DSA, RSA, Merkle trees [34] and BGLS [8]) are explored as potential alternatives for data authentication primitives. In [30, 45] *verification objects* VO are deployed to authenticate data retrieval in "edge computing" In [42, 44] Merkle tree and cryptographic hashing constructs are deployed to authenticate range query results.

To summarize, existing secure outsourced query mechanisms deploy (i) partitioning-based schemes and symmetric key encryption for ("statistical" only) confidentiality, (ii) homomorphisms for oblivious aggregation (SUM, COUNT) queries (simply too slow to be practical), (iii) hash trees/chains and (iv) signature chaining and aggregation to ensure correctness of selection/range queries and projection operators. SUM, COUNT, and projection usually behave linearly in the database size. Selection and range queries may be performed in constant time, logarithmic time or linear time depending on the queried attribute (e.g., whether it is a primary key) and the type of index used.

For illustration purposes, w.l.o.g., consider a scenario most favorable to outsourcing, i.e., assuming the operations behave linearly and are extremely selective, only incurring two 32-bit data transfers between the client and the cloud (one for the instruction and one for the result). Informally, to offset the network cost of $900 \times 32 \times 2 = 57,600$ picocents, only traversing a database of size at least 10^5 will generate enough CPU cycle cost savings. Thus it seems that with very selective queries (returning very little data) over large enough databases, outsourcing can break even.

Cost of Security. In the absence of security constructs, we were able to build a scenario for which outsourcing is viable. But what about a general scenario? What are the overheads of security there? It is important to understand whether the cost savings will be enough to offset them. While detailing individual secure query protocols is out of scope here, it is possible to reason generally and gain an insight into the associated order of magnitudes.

Existing integrity mechanisms deploy hash trees, hash chains and signatures to secure simple selection, projection or range queries. Security overheads would then include *at least* the (client-side) hash tree proof re-construction ($O(\log n)$ crypto-hashes) and subsequent signature verification of the tree's root. The hash tree proofs are often used to authenticate range boundaries. The returned element set is then authenticated often through either a hash chain (in the case of range joins, at least 30 picocents per byte) or aggregated signature constructs (e.g., roughly 60,000 picocents each, for selects or projections). This involves either modular arithmetic or crypto-hashing of the order of the result data set. For illustration purposes, we will again favor the case for outsourcing, and assume only crypto-hashing and a linear operation are applied.

Consider a database that has $n = 10^9$ tuples of 64 bits each. In that case (binary) hash tree nodes need to be at least 240 bits (80 + 160 bits = 2 pointers + hash value) long. If we assume 3 CPU cycles are needed per data item, the boundary condition results in selectivity $s \leq 0.00037$ before outsourcing starts to make economical sense. In a more typical scenario of $s = 0.001$ (queries are returning 0.1% of the tuples), a per-query loss of over 0.3 US cents will be incurred.

The above holds only for the $S \rightarrow L$ scenario in which hash trees are deployed. In the case of signature aggregation [41, 42], the break-even selectivity would be even lower due to the higher computation overheads.

5 To Conclude

In this paper we explored whether cryptography can be deployed to secure cloud computing against insiders. We estimated common cryptography costs (AES, MD5, SHA-1, RSA, DSA, and ECDSA) and finally explored outsourcing of data and computation to untrusted clouds. We showed that deploying the cloud as a simple remote encrypted file system is extremely unfeasible if considering only core technology costs. We also concluded that existing secure outsourced data query mechanisms are mostly cost-unfeasible because **today's cryptography simply lacks the expressive power to efficiently support outsourcing** to untrusted clouds. Hope is not lost however. We found borderline cases where outsourcing of simple range queries can break even when compared with local execution. These scenarios involve large amounts of outsourced data (e.g., 10^9 tuples) and extremely selective queries which return only an infinitesimal fraction of the original data (e.g., 0.00037%).

References

[1] IBM 4764 PCI-X Cryptographic Coprocessor. Online at `http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml`, 2007.

[2] E. I. Administration. "average retail price of electricity to ultimate customers by end-use sector, by state". Online at `http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html`.

[3] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST 07)*, Berkeley, CA, USA, 2007. USENIX Association.

[4] G. Amanatidis, A. Boldyreva, and A. O'Neill. Provably-secure schemes for basic query support in outsourced databases. In S. Barker and G.-J. Ahn, editors, *DBSec*, volume 4602 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2007.

[5] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2007.

[6] D. J. Bernstein and T. L. (editors). ebacs: Ecrypt benchmarking of cryptographic systems. Online at `http://bench.cr.yp.to` accessed 30 Jan. 2009.

[7] M. Blaze. A Cryptographic File System for Unix. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 9–16, Fairfax, VA, 1993. ACM.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EuroCrypt*, 2003.

[9] G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano. The Design and Implementation of a Transparent Cryptographic Filesystem for UNIX. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 245–252, Boston, MA, June 2001.

[10] Y. Chen and R. Sion. To cloud or not to cloud?: musings on costs and viability. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 29:1–29:7, New York, NY, USA, 2011. ACM.

[11] CNN. Feds seek Google records in porn probe. Online at `http://www.cnn.com`, Jan. 2006.

[12] CNN. YouTube ordered to reveal its viewers. Online at `http://www.cnn.com`, July 2008.

[13] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, New York, NY, USA, 2006. ACM.

[14] P. T. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *IFIP Workshop on Database Security*, pages 101–112, 2000.

[15] Donna Bogatin. Google Apps data risks: Security vs. privacy. Online at `http://blogs.zdnet.`

com/micro-markets/?p=1021, Feb. 2007.

[16] J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 59–70. ACM New York, NY, USA, 1999.

[17] J. Fetzer. Internet data centers:end user & developer requirements. Online at http://www.utilityeda.com/Summer2006/Mares.pdf.

[18] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, Bolton Landing, NY, October 2003. ACM SIGOPS.

[19] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. In *SIGCOM Computer Communications Review*, 2009.

[20] T. G. Grid. Green grid metrics: Describing data center power efficiency. Online at http://www.thegreengrid.org/gg_content/Green_Grid_Metrics_WP.pdf.

[21] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 216–227. ACM Press, 2002.

[22] J. Hamilton. Internet-scale service efficiency. Large Scale Distributed Systems & Middleware (LADIS 2008),, 2008.

[23] J. Hamilton. On designing and deploying internet-scale services. Technical report, Windows Live Services Platform, Microsoft, 2008.

[24] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of ACM SIGMOD*, 2004.

[25] IBM. IBM blade servers. Online at http://www-03.ibm.com/systems/bladecenter/hardware/servers/.

[26] S. H. John Markoff. Hiding in plain sight, google seeks more power. Online at http://www.nytimes.com/2006/06/14/technology/14search.html.

[27] A. Kashyap, S. Patil, G. Sivathanu, and E. Zadok. I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, pages 69–79, Atlanta, GA, November 2004. USENIX Association.

[28] R. Lab. How fast is the RSA algorithm? Online at http://www.rsa.com/rsalabs/node.asp?id=2215.

[29] Larry Dignan. Will you trust Google with your data? Online at http://blogs.zdnet.com/BTL/?p=4544, Feb. 2007.

[30] M. Atallah and C. YounSun and A. Kundu. Efficient Data Authentication in an Environment of Untrusted Third-Party Distributors. In *24th International Conference on Data Engineering ICDE*, pages 696–704, 2008.

[31] Maithili Narasimha and Gene Tsudik. DSAC: integrity for outsourced databases with signature aggregation and chaining. Technical report, 2005.

[32] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authenticated data structures. Technical report, 2001.

[33] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[34] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.

[35] J. Merritt. What google searches and data mining mean for you. Online at http://www.talkleft.com/story/2006/01/25/692/74066.

[36] C. Metz. Google demanding intel's hottest chips? Online at http://www.theregister.co.uk/2008/10/15/google_and_intel/.

[37] Microsoft Research. Encrypting File System for Windows 2000. Technical report, Microsoft Corporation, July 1999. www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp.

[38] R. Miller. Microsoft: Pue of 1.22 for data center containers. Online at http://www.datacenterknowledge.com/archives/2008/10/20/microsoft-pue-of-122-for-data-center-containers/.

[39] A.-F. Mohammad, L. Alexander, and V. Amin. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[40] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *Proceedings of Network and Distributed System Security (NDSS)*, 2004.

[41] E. Mykletun, M. Narasimha, and G. Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In *Computer Security - ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2004.

[42] M. Narasimha and G. Tsudik. Authentication of Outsourced Databases using Signature Aggregation and Chaining. In *Proceedings of DASFAA*, 2006.

[43] Optimum. Optimum online plans. Online at http://www.buyoptimum.com.

[44] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *Proceedings of ACM SIGMOD*, 2005.

[45] H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 560, Washington, DC, USA, 2004. IEEE Computer Society.

[46] G. Sivathanu, C. P. Wright, and E. Zadok. Enhancing File System Integrity Through Checksums. Technical Report FSL-04-04, Computer Science Department, Stony Brook University, May 2004. www.fsl.cs.sunysb.edu/docs/nc-checksum-tr/nc-checksum.pdf.

[47] Tingjian Ge and Stan Zdonik. Answering aggregation queries in a secure system model. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 519–530. VLDB Endowment, 2007.

[48] Whitfield Diffie. How Secure Is Cloud Computing? Online at http://www.technologyreview.com/computing/23951/, Nov. 2009.

Privacy-Preserving Fine-Grained Access Control in Public Clouds

Mohamed Nabeel, Elisa Bertino
{nabeel, bertino}@cs.purdue.edu
Department of Computer Science and Cyber Center
Purdue University

Abstract

With many economical benefits of cloud computing, many organizations have been considering moving their information systems to the cloud. However, an important problem in public clouds is how to selectively share data based on fine-grained attribute based access control policies while at the same time assuring confidentiality of the data and preserving the privacy of users from the cloud. In this article, we briefly discuss the drawbacks of approaches based on well known cryptographic techniques in addressing such problem and then present two approaches that address these drawbacks with different trade-offs.

1 Introduction

With the advent of technologies such as cloud computing, sharing data through a third-party cloud service provider has never been more economical and easier than now. However, such cloud providers cannot be trusted to protect the confidentiality of the data. In fact, data privacy and security issues have been major concerns for many organizations utilizing such services. Data often contains sensitive information and should be protected as mandated by various organizational policies and legal regulations. Encryption is a commonly adopted approach to assure data confidentiality. Encryption alone however is not sufficient as organizations often have also to enforce fine-grained access control on the data. Such control is often based on security-relevant properties of users, referred to as *identity attributes*, such as the roles of users in the organization, projects on which users are working, and so forth. These access control systems are referred to as *attribute based access control (ABAC) systems*. Therefore, an important requirement is to support fine-grained access control, based on policies specified using identity attributes, over encrypted data.

With the involvement of the third-party cloud services, a crucial issue is that the identity attributes in the access control policies may reveal privacy-sensitive information about users and organizations and leak confidential information about the content. The confidentiality of the content and the privacy of the users are thus not assured if the identity attributes are not protected. It is well-known that privacy, both individual as well as organizational, is considered a key requirement in all solutions, including cloud services, for digital identity management [7]. Further, as insider threats are one of the major sources of data theft and privacy breaches, identity attributes must be strongly protected even from accesses within organizations. With initiatives such as cloud computing the scope of insider threats is no longer limited to the organizational

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

perimeter. Therefore, protecting the identity attributes of the users while enforcing attribute-based access control both within the organization as well as in the cloud is crucial.

For example, let us consider a hospital that decides to use the cloud to manage their electronic health record (EHR) system. Since EHRs are sensitive information, their confidentiality should be preserved from the cloud. A typical hospital stakeholders consist of employees playing different roles such as receptionist, cashier, doctor, nurse, pharmacist, system administrator, and so on. A cashier, for example, does not need have access to data in EHRs except the billing information in them while a doctor or a nurse does not need have access to billing information in EHRs. This requires the cloud based EHR system to support fine-grained access control. The typical identity attributes used by the stakeholders in our EHR system, such as role, location and position, can be used as good contextual information to connect with other publicly available information in order to learn sensitive information about individuals, leading to privacy violations. For example, if system administrators of the EHR system can see hospital employees' identity attributes, they can misuse the system to access EHRs and sell to outsiders without being caught. In order to address these issues, the cloud based EHR system should protect the identity attributes of users.

The goal of this article is to provide an overview of our approaches to enforce fine-grained access control on sensitive data stored in untrusted public clouds, while at the same assuring the confidentiality of the data from the cloud and preserving the privacy of users who are authorized to access the data. We compare these approaches and discuss about open issues.

The article is organized as follows. Section 2 briefly discusses the drawbacks of existing cryptographic techniques and presents a new approach for managing group encryption keys. Based on such new key management approach, Sections 3 and 4 present a basic approach and a two layer encryption-based approach for privacy-preserving ABAC for data on clouds, respectively. Section 5 compares the existing and new approaches. Finally, Section 3 outlines a few conclusions.

2 A New Approach to Manage Group Encryption Keys

An approach to support fine-grained selective ABAC is to identify the sets of data items to which the same access control policy (or set of policies) applies and then encrypt each such set with the same encryption key. The encrypted data is then uploaded to the cloud and each user is given the keys only for the set(s) of data items that it can access according to the policies [1]. Such approach addresses two requirements: (a) protecting data confidentiality from the cloud; (b) enforcing fine-grained access control policies with respect to the data users. A major issue in such an approach is represented by key management, as each user must be given the correct keys with respect to the access control policies that the user satisfies. One approach to such issue is to use a hybrid solution whereby the data encryption keys are encrypted using a public key cryptosystem such as attribute based encryption (ABE) [1] and/or proxy re-encryption (PRE) [10] [11]. However, such an approach has several weaknesses: it cannot efficiently handle adding/revoking users or identity attributes, and policy changes; it requires to keep multiple encrypted copies of the same key; it incurs high computational costs; it requires additional attributes to support revocation [2]. Therefore, a different approach is required.

It is also worth noting that a simplistic group key management (GKM) scheme by which the content publisher directly delivers the symmetric keys to the corresponding users has some major drawbacks with respect to user privacy and key management. On one hand, user private information encoded in the user identity attributes is not protected in the simplistic approach. On the other hand, such a simplistic key management scheme does not scale well when the number of users becomes large and multiple keys need to be distributed to multiple users. The goal of our work is to develop an approach which does not have these shortcomings.

[1] Here and the rest of the article the term user may indicate a real human user or some client software running on behalf of some human user.

We observe that, without utilizing public key cryptography and by allowing users to dynamically derive the symmetric keys at the time of decryption, one can address the above issues. Based on this idea, we have defined a new GKM scheme, called broadcast GKM (BGKM), and given a secure construction of the BGKM scheme [6]. The idea is to give secrets to users based on the identity attributes they have and later allow them to derive actual symmetric keys based on their secrets and some public information. A key advantage of the BGKM scheme is that adding users/revoking users or updating access control policies can be performed efficiently and only requires updating the public information. Our BGKM scheme is referred to as access control vector BGKM (ACV-BGKM). The idea of ACV-BGKM is to construct a special matrix A where each row is linearly independent and generated using each user's secret. The group controller generates the null space Y of this matrix by solving the linear system $AY = 0$, randomly selects a vector in the null space, and hides the group symmetric key inside this vector. We call this vector as access control vector (ACV) and is part of the public information. An authorized user can generate a vector in the row space of the special matrix using its secret and some public information. We call this vector as key extraction vector (KEV). The system is designed such that the inner product of ACV and KEV allows authorized users to derive the group symmetric key. We show that a user who does not have a valid secret has a negligible probability of generating a valid KEV and deriving the group key. When a user is revoked, the group controller simply updates the special matrix excluding the revoked user and generate a new ACV hiding a new group key. Notice that such revocation handling does not affect the existing users as only the public information is changed.

Using the ACV-BGKM scheme as a building block, we have constructed a more expressive scheme called attribute based GKM (AB-GKM) [5]. The idea is to generate an ACV-BGKM instance for each attribute and combine the instances together using an access structure that represents the attribute based access control policy. The AB-GKM scheme satisfies all the properties of the ACV-BGKM scheme and consists of the following five algorithms: **Setup**, **SecGen**, **KeyGen**, **KeyDer**and **Update**.

- **Setup**(ℓ): It initializes the BGKM scheme using a security parameter ℓ. It also initializes the set of used secrets **S**, the secret space $\mathcal{SS}$, and the key space $\mathcal{KS}$.

- **SecGen(user, attribute)**: It picks a random bit string $s \notin$ **S** uniformly at random from $\mathcal{SS}$, adds s to **S** and outputs s. A unique secret is assigned to per user per attribute. These secrets are used by the group controller to generate the group key and also by the users to derive the group key.

- **KeyGen(S, Policy)**: It picks a group key k uniformly at random from the key space $\mathcal{KS}$ and outputs the public information tuple PI computed from the secrets in **S** that satisfy the policy and the group key k. PI indirectly encodes the policy such that a user can use PI along with its secrets only if the user satisfies the policy used to generate PI.

- **KeyDer**(s, PI): It takes the user's secret s and the public information PI to output the group key. The derived group key is equal to k if and only if $s \in$ **S** and satisfies the policy.

- **Update(S)**: Whenever the set **S** changes, a new group key k' is generated. Depending on the construction, it either executes the **KeyGen** algorithm again or incrementally updates the output of the last **KeyGen** algorithm.

3 Basic Approach to Privacy-Preserving ABAC

Using our AB-BGKM scheme, we have developed an ABAC mechanism whereby a user is able to decrypt the data if and only if its identity attributes satisfy the data owner's policies, whereas the data owner and the cloud learn nothing about user's identity attributes. The mechanism is fine-grained in that different policies

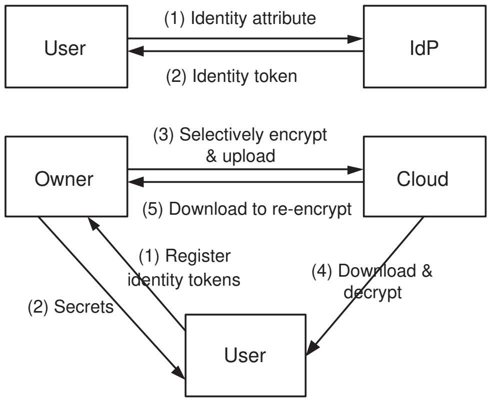


Figure 1: Overall System Architecture

can be associated with different sets of data items. A user can derive only the encryption keys associated with the sets of data items the user is entitled to access.

We now give an overview of the overall scheme. As shown in Figure 1, our scheme for policy based content sharing in the cloud involves four main entities: the *Data Owner* (Owner), the *Users* (Usrs), the *Identity Providers* (IdPs), and the *Cloud Storage Service* (Cloud). Our approach is based on three main phases: *identity token issuance*, *identity token registration*, and *data management*.

Identity Token Issuance

IdPs issue *identity tokens* for certified identity attributes to Usrs. An identity token is a Usr's identity encoded in a specified electronic format in which the involved identity attribute value is represented by a semantically secure cryptographic *commitment*.[2] We use the Pedersen commitment scheme [8]. Identity tokens are used by Usrs during the identity token registration phase.

Identity Token Registration

In order to be able to decrypt the data to be downloaded from the Cloud, Usrs have to register at the Owner. During the registration, each Usr presents its identity tokens and receives from the Owner a set of secrets for each identity attribute based on the **SecGen** algorithm of the AB-GKM scheme. These secrets are later used by Usrs to derive the keys to decrypt the sets of data items for which they satisfy the access control policy using the **KeyDer** algorithm of the AB-GKM scheme. The Owner delivers the secrets to the Usrs using a privacy-preserving approach based on the OCBE protocols [4]. The OCBE protocols ensure that a Usr can obtain the secrets if and only if the Usr's committed identity attribute value (within Usr's identity token) satisfies the matching condition in the Owner's access control policy, while the Owner learns nothing about the identity attribute value. Note that not only the Owner does not learn anything about the actual value of Usrs' identity attributes but it also does not learn which policy conditions are verified by which Usrs, thus the Owner cannot infer the values of Usrs' identity attributes.

Data Management

The Owner groups the access control policies into *policy configurations*. The data items are partitioned into sets of data items based on the access control policies. The Owner generates the keys based on the access control policies in each policy configuration using the **KeyGen** algorithm of the AB-GKM scheme and selectively encrypts the different data item sets. These encrypted data item sets are then uploaded to the Cloud. Usrs download encrypted data item sets from the Cloud. The **KeyDer** algorithm of the AB-GKM

[2]A cryptographic commitment allows a user to commit to a value while keeping it hidden and preserving the user's ability to reveal the committed value later.

scheme allows Usrs to derive the key K for a given policy configuration using their secrets in an efficient and secure manner. With this scheme, our approach efficiently handles new Usrs and revocations to provide forward and backward secrecy. The system design also ensures that access control policies can be flexibly updated and enforced by the Owner without changing any information given to Usrs.

An Example

Using the same EHR system presented earlier, we now provide an example showing the data management phase.

The data items of an EHR such as Contact Info, Lab Report and Clinical Record are allowed to be accessed by different employees based on their roles and other identity attributes. Suppose the roles for the hospital's employees are: receptionist (rec), cashier (cas), doctor (doc), nurse (nur), data analyst (dat), and pharmacist (pha). Three selected ACPs of the EHR system are shown below.

1. $ACP_1 = (\text{"}\mathsf{role} = \text{rec"}, \{\langle\text{ContactInfo}\rangle\})$
2. $ACP_2 = (\text{"}\mathsf{role} = \text{doc"}, \{\langle\text{ClinicalRecord}\rangle\})$
3. $ACP_3 = (\text{"}\mathsf{role} = \text{nur} \wedge \mathsf{level} \geq 5\text{"}, \{\langle\text{ContactInfo}\rangle, \langle\text{ClinicalRecord}\rangle\})$

The first ACP says that a receptionist can access Contact Info data items, the second says that a doctor can access Clinical Record data items, and the third says that a nurse with level greater than or equal to 5 can access Contact Info and Clinical Record data items. Based on these policies the hospital identifies the policy configuration for each set of data items. For the above sample policies, the hospital creates two policy configurations as follows:

1. $\mathsf{PC}_1 = (\langle\text{ContactInfo}\rangle\text{: } ACP_1, ACP_3)$
2. $\mathsf{PC}_2 = (\langle\text{ClinicalRecord}\rangle\text{: } ACP_2, ACP_3)$

The hospital generates a group key and public information using the **KeyGen** algorithm for PC_1, and encrypts Contact Info data items using the group key. Hospital employees who satisfy either ACP_1 or ACP_3 in PC_1 can derive the group using the **KeyDer** algorithm and decrypt the Contact Info data items downloaded from the cloud. A similar process is followed for PC_2 and other policy configurations not shown in the example.

3.1 Implementation and Security/Performance Analysis

We have implemented our basic approach on Amazon S3 which is a popular cloud based storage service. The content management consists of two tasks. First, the Owner encrypts the data item sets based on the access control policies and uploads the encrypted sets along with some meta-data. Then, authorized users download the encrypted data items sets and meta-data from the Cloud, and decrypt the data item sets using the secrets they have.

Now we illustrate the interactions of the Owner with Amazon S3 as the Cloud. In our implementation, we have used the REST API to communicate with Amazon S3. Figure 2 shows the overall involvement of the Owner in the user and content management process when uploading the data item sets to Amazon S3. While the fine-grained access control is enforced by encrypting using the keys generated through the AB-GKM scheme, it is important to limit the access to even the encrypted data item sets in order to minimize the bandwidth utilization. We associate a hash-based message authentication code (HMAC) with each encrypted data item sets such that only the users having valid identity attributes can produce matching HMACs.

Initially the Owner creates a *bucket*, which is a logical container in S3, to store encrypted data item sets as *objects*. Subsequently, the Owner executes the following steps:

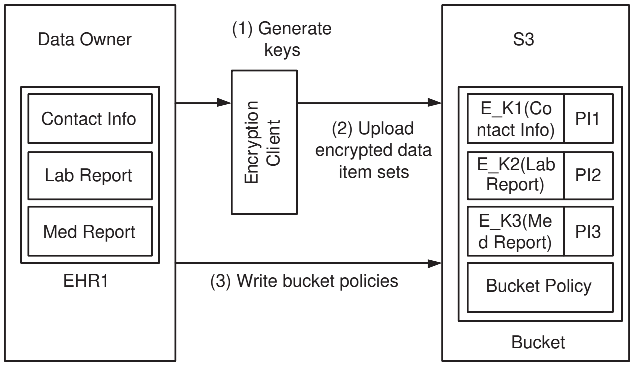


Figure 2: Overall involvement of the Owner

1. The Owner generates the symmetric keys using the AB-GKM's **KeyGen** algorithm and instantiates an encryption client. Note that the Owner generates a unique symmetric key for each policy configuration.

2. Using the encryption client as a proxy, the Owner encrypts the data item sets and uploads the encrypted data item sets along with the public information needed to generate the keys as the meta-data.

3. The Owner generates HMACs using the symmetric keys and PIs. These HMACs are used to write *bucket policies* that control access to encrypted data item sets. The bucket policies define access rights for the encrypted data item sets. It should be noted that only the Owner has access to the bucket policies. While S3 has access to the HMACs, by just knowing the HMACs S3 is not able to decrypt the data.

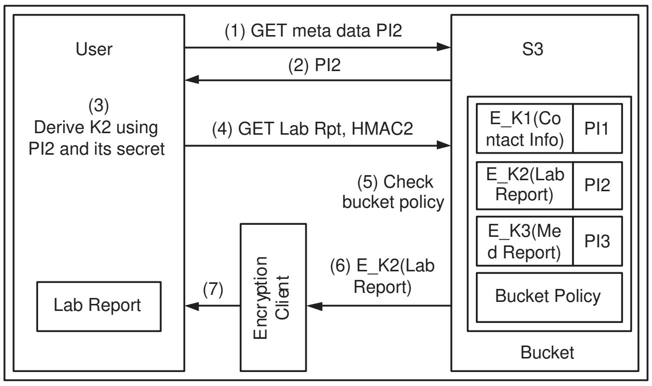


Figure 3: Downloading data item sets from Amazon S3 as the Cloud

Now we look at users' involvement in content management. Figure 3 shows the overall involvement of users with Amazon S3 when downloading encrypted sets of data items. The following steps are involved:

1. Users are allowed to download the meta-data for any encrypted data item set.

2. A user downloads the public information associated with the data item set it wants to access.

3. Using the **KeyDer** algorithm of the AB-GKM scheme, the user derives the symmetric key. Notice that the user can derive a valid symmetric only if its identity attributes satisfy the access control policy associated with the data item set.

4. Using the public information and the derived symmetric key, the user creates an HMAC and submits it to S3.

5. S3 checks if the HMAC in the bucket policy matches with the one the user submitted. If it does not match, it denies access.

6. The user downloads the encrypted data item set.

7. The encryption client decrypts the data item set.

3.2 Performance Enhancements

In this section, we discuss some enhancements to improve the performance of the basic ACV-BGKM scheme [9], which is the underlying construct of the AB-GKM scheme,by using two techniques: bucketization and subset cover.

3.2.1 Bucketization

Our ACV-BGKM scheme works efficiently even when there are thousands of Usrs. However, as the upper bound n of the number of involved Usrs gets large, solving the linear system $AY = 0$ over a large finite field $\mathbb{F}_q$, which the key operation in the **KeyGen** algorithm, becomes the most computationally expensive operation in our scheme. Solving this linear system with the method of Gaussian-Jordan elimination takes $O(n^3)$ time. Although this computation is executed at the data owner, which is usually capable of carrying on computationally expensive operations, when n is very large, e.g., $n = 100,000$, the resulting costs may be too high for the data owner. Due to the non-linear cost associated with solving a linear system, we can reduce the overall computational cost by breaking the linear system in to a set of smaller linear systems. We have thus defined a two-level approach. Under this approach, the data owner divides all the involved Usrs into multiple "buckets" (say m) of a suitable size (e.g., 1000 each), computes an intermediate key for each bucket by executing the **KeyGen** algorithm, and then computes the actual group key for all the Usrs by executing the **KeyGen** algorithm with the intermediate keys as the secrets. Note that the intermediate key generation can be parallelized as each bucket is independent. The data owner executes $m + 1$ **KeyGen** algorithms of smaller size. The complexity of the **KeyGen** algorithm is proportional to $O(n^3/m^2 + m^3)$. It can be shown that the optimal solution is achieved when m reaches close to $n^{3/5}$.

3.2.2 Subset Cover

The bucketization approach becomes inefficient when the bucket size increases. The issue is that the bucketization still utilizes the basic ACV-BGKM scheme. In our basic ACV-BGKM scheme, as each Usr is given a single secret, the complexity of PI and all algorithms is proportional to n, that is, the number of Usrs in the group. We have thus defined an approach based on a technique from previous research on broadcast encryption [3] to improve the complexity to sub-linear in n. Based on such technique, one can make the complexity sub-linear in the number of Usrs by giving more than one secret during **SecGen** for each attribute Usrs possess. The secrets given to each Usr overlaps with different subsets of Usrs. During the **KeyGen**, the data owner identifies the minimum number of subsets to which all the Usrs belong and uses one secret per the identified subset. During **KeyDer**, a Usr identifies the subset it belongs to and uses the corresponding secret to derive the group key. Group dynamics are handled by making some of the secrets given to Usrs invalid.

4 Two-layer Encryption Approach to Privacy-Preserving ABAC

Our basic approach follows the conventional data outsourcing scenario where the Owner enforces *all* the access control policies through selective encryption and uploads encrypted data to the untrusted Cloud. We refer to this approach as single layer encryption (SLE). The SLE approach supports fine-grained attribute-based access control policies and preserves the privacy of users from the Cloud. However, in such an approach, the Owner is in charge of encrypting the data before uploading it to the third-party server as well as re-encrypting the data whenever user credentials or authorization policies change and managing the encryption keys. The Owner has to download all affected data before before performing the selective encryption. The Owner thus incurs high communication and computation costs, which then negate the benefits of using a third party service. A better approach should delegate the enforcement of fine-grained access control to the Cloud, so to minimize the overhead at the Owner, whereas at the same time assuring data confidentiality from the third-party server.

In this section, we provide an overview of an approach, based on two layers of encryption, that addresses such requirement. Under such approach, referred to as *two-layer encryption* (TLE), the Owner performs a coarse grained encryption, whereas the Cloud performs a fine grained encryption on top of the data encrypted by the coarse grained encryption. A challenging issue in this approach is how to decompose the ABAC policies such that the two-layer encryption can be performed. In order to delegate as much access control enforcement as possible to the Cloud, one needs to decompose the ABAC policies so that the Owner only needs to manage the minimum number of attribute conditions in these policies that assures the confidentiality of data from the Cloud. Each policy should be decomposed into two subpolicies such that the conjunction of the two subpolicies result in the original policy. The two-layer encryption should be performed such that the Owner first encrypts the data based on one set of subpolicies and the Cloud re-encrypts the encrypted data using the other set of policies. The two encryptions together enforce the original policies as users should perform two decryptions in order to access the data. For example, consider the policy $(C_1 \wedge C_2) \vee (C_1 \wedge C_3)$. This policy can be decomposed as two subpolicies C_1 and $C_2 \vee C_3$. Notice that the decomposition is consistent; that is, $(C_1 \wedge C_2) \vee (C_1 \wedge C_3) = C_1 \wedge (C_2 \vee C_3)$. The Owner enforces the former by encrypting the data for the users satisfying the former and the Cloud enforces the latter by re-encrypting the Owner encrypted data for the users satisfying the latter. Since the Cloud does not handle C_1, it cannot decrypt the Owner encrypted data and thus confidentiality is preserved. Notice that users should satisfy the original policy to access the data by performing two decryptions. An analysis of this approach suggests that the problem of decomposing for coarse and fine grained encryption while assuring the confidentiality of data from the third party and the two encryptions together enforcing the policies is NP-complete. We have thus investigated optimization algorithms to construct near optimal solutions to this problem. Under our TLE approach, the third party server supports two services: the storage service, which stores encrypted data, and the access control service, which performs the fine grained encryption.

As shown in Figure 4, we utilize the same AB-GKM scheme that allows users whose attributes satisfy a certain policy to derive the group key and decrypt the content they are allowed to access from the Cloud. Our proposed approach assures the confidentiality of the data and preserves the privacy of users from the access control service as well as the cloud storage service while delegating as much of the access control enforcement as possible to the third party through the two- layer encryption technique.

The TLE approach has many advantages. When the policy or user dynamics changes, only the outer layer of the encryption needs to be updated. Since the outer layer encryption is performed at the third party, no data transmission is required between the Owner and the third party. Further, both the Owner and the third party service utilize the AB-GKM scheme for key management whereby the actual keys do not need to be distributed to the users. Instead, users are given one or more secrets which allow them to derive the actual symmetric keys for decrypting the data.

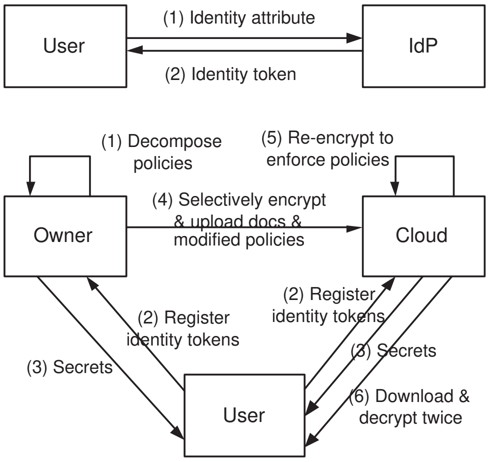


Figure 4: Two Layer Encryption Approach

5 Comparison of Approaches

In this section we compare ABE-based existing approaches as a whole and the two AB-GKM based approaches presented earlier. A common characteristic of all these approaches is that they support secure attribute based group communication.

Table 1: Comparison of Approaches

Property	**ABE**	**SLE**	**TLE**
Cryptosystem	Asymmetric	Symmetric	Symmetric
Secure attribute based group communication	Yes	Yes	Yes
Efficient revocation	No	Yes	Yes
Delegation of access control	No	No	Yes

As shown in Table 1, while ABE-based approaches rely on asymmetric cryptography, our two approaches rely only on symmetric cryptography which is more efficient than the asymmetric cryptography. A key issue in the ABE-based approaches is that they do not support efficient user revocations unless they use additional attributes [2]. Our schemes address the revocation issue. It should be noted that the ABE based approaches and our SLE approach follows the conventional data outsourcing scenario by which the data owner manages all users and data before uploading the encrypted data to the cloud, whereas the TLE based approach provides the advantage of partial management of users and data in the cloud itself while assuring confidentiality of the data and privacy of users. With ever increasing user base and large amount of data, while such delegation of user management and access control is becoming very important, it also has trade offs in terms of privacy. Compared to the SLE approach, in the TLE approach, the data owner has to reveal partial access control policies to the cloud which may allow the cloud to infer some details about the identity attributes of users. It is an interesting topic to investigate how to construct symmetric key based practical solutions to hide the access control policies from the cloud while utilizing the benefits of delegation of control.

6 Conclusions

Current trends in computing infrastructures like Service Oriented Architectures (SOAs) and cloud computing are further pushing publishing functions to third-party providers to achieve economies of scale. However, recent surveys by IEEE and Cloud Security Alliance (CSA) have found that one of the key resistance factor for companies and institutions to move to the cloud is represented by data privacy and security concerns. Our AB-GKM based approaches address such privacy and security concerns in the context of efficient and flexible sharing and management of sensitive content. Compared to state of the art ABE based approaches, our approaches support efficient revocation and management of users which is a key requirement to construct scalable solutions.

References

[1] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *SP 2007: Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[2] J. Camenisch, M. Dubovitskaya, R. R. Enderlein, and G. Neven. Oblivious transfer with hidden access control from attribute-based encryption. In *SCN 2012: Proceedings of the 8th International Conference on Security and Cryptography for Networks*, pages 559–579, 2012.

[3] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *CRYPTO 2001: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 47–60, 2002.

[4] J. Li and N. Li. OACerts: Oblivious attribute certificates. *IEEE Transactions on Dependable and Secure Computing*, 3(4):340–352, 2006.

[5] M. Nabeel and E. Bertino. Towards attribute based group key management. In *CCS 2011: Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[6] M. Nabeel, N. Shang, and E. Bertino. Privacy preserving policy based content sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2012.

[7] OpenID. `http://openid.net/` [Last accessed: Oct. 14, 2012].

[8] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 129–140, 1992.

[9] N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *ICDE 2010: Proceedings of the 2010 IEEE 26th International Conference on Data Engineering*, 2010.

[10] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *ASIACCS 2010: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270, 2010.

[11] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM 2010: Proceedings of the 29th conference on Information communications*, pages 534–542, 2010.

The Blind Enforcer: On Fine-Grained Access Control Enforcement on Untrusted Clouds*

Dinh Tien Tuan Anh, Anwitaman Datta
School of Computer Engineering, Nanyang Technological University, Singapore
{ttadinh, anwitaman}@ntu.edu.sg

Abstract

Migration of one's computing infrastructure to the cloud is gathering momentum with the emergence of relatively mature cloud computing technologies. As data and computation are being outsourced, concerns over data security (such as confidentiality, privacy and integrity) remain one of the greatest hurdles to overcome. In the meanwhile, the increasing need for sharing data between or within cloud-based systems (for instance, sharing between enterprise systems or users of a social network application) demands even more care in ensuring data security. In this paper, we investigate the challenges in outsourcing access control of user data to the cloud. We identify what constitute a fine-grained cloud-based access control system and present the design-space along with a discussion on the current state-of-the-art. We then describe a system which extends an Attribute-Based Encryption scheme to achieve more fine-grainedness as compared to existing approaches. Our system not only protects data from both the cloud service provider and unauthorized access from other users, it also moves the heavy computations towards the cloud, taking advantage of the latter's relatively unbounded resources. Additionally, we integrate an XML-based framework (XACML) for flexible, high-level policy management. Finally, we discuss some open problems, solving which would lead to a further robust and flexible cloud-based access control system.

Keywords: *fine-grained access control, Attribute-Based Encryption, XACML, untrusted cloud*

1 Introduction

An enormous amount of data is continuously being generated, with sources ranging from traditional enterprise systems, to social, Web 2.0 applications. It is becoming increasingly common to process data continuously in almost real time as it arrives in streams, in addition to processing (archived) data in batches. Examples of enterprise-scale systems that generate data from their own dedicated sensing infrastructure include stock monitoring,[1] meteorological and environmental monitoring[2] and traffic monitoring [5]. The increased popularity of social and Web 2.0 applications, especially social computing applications like YouTube, Facebook, Twitter, Wikipedia accounts for ever-increasing streams of user-generated content. Finally, ubiquitous

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work is supported by A*Star grant 102 158 0038.
[1] `www.xignite.com`
[2] `www.noaa.gov`

devices such as smart phones equipped with sensing capabilities are driving the growth of other types of social applications such as participatory sensing [15] and personal health monitoring [3].environmental and physiological data.

The variety and abundance of data, coupled with the potential of social interactivity and mash-up services bring sharing of the data to the foreground. For enterprise systems, sharing may translate to new business opportunities, from real-time decision making to smart-city solutions. For social applications, sharing facilitates conformation with the norms or enforcing social bonds. Social computing paradigms such as crowd-sourcing and participatory sensing provide cost-effective alternative services to those using dedicated infrastructure (even when the latter is economically viable).

A critical problem with sharing data is the myriad of security implications. data security has concerned primarily with the question of who gets access to which data. However, the focus has shifted to the question of what aspects of the data to share, and under what context the data can be used. The latter is related to *data privacy*. In this paper, we will be focusing on *fine-grained access control*. The increasing volume of structured data (often arriving in stream) and of the applications (social contexts, near real-time decision support, mash-ups, etc.) demand a scalable and finer access control than the simple all-or-nothing binary sharing. In the following, we identify a core list of access control policies which are found useful in many applications (more complex access scenarios can be derived by combining these *primitives*). For simplicity, let us assume the data consists of multiple attributes, each attribute domain being an ordered space.

- **Filtering policy:** grants access to an attribute if a function is evaluated to `true`. For example, to help detect financial fraud, multiple banks may need to share data of their clients' transactions which contain sensitive information. To achieve this, each bank may only need to share the IDs of *abnormal* transactions whose values exceed a certain threshold. Thus, the function would be $f := \texttt{transaction\_value} \geq \theta$ for a threshold value θ.

- **Granularity policy:** grants access to a *noisy* version of an attribute. For example, in a location sharing application, Alice may not want Bob to know her exact location, but reveal only the area she is in (neighborhood, county, state or country). The policy specifies a granularity level g and a function which transforms the data to different levels of granularity.

- **Similarity policy:** grants access to an user if he can provide inputs that are *similar* to the attribute being shared. For instance, in an online trading system, a selling user submits a price for her item and a set of buyers submitting their estimated price for the item. The seller may want to reveal her price only if it is close to the buyer's estimation. This policy must specify a *closeness* distance within which the attribute will be shared.

- **Summary policy:** grants access to the aggregates (or max, min values) over windows of data. For instance, a mobile health application collects user vital sign (heart rate, blood pressure) every minute. An user wishing to share her data for research purposes may want to reveal only the average, maximum and minimum readings per day (or per sliding window of 24 data items). This policy requires a summary function and specification of a sliding window (starting point, window size and advance step).

Enforcing such access control over a proprietary infrastructure is relatively straightforward. However, instead of building a computing infrastructure from scratch, one can now enjoy the instantly available, elastic and virtually unbounded resources offered by cloud computing vendors at competitive prices [4]. Many

[3] `www.zephyr-technology.com`

[4] `https://developers.google.com/appengine` & `http://aws.amazon.com/ec2`

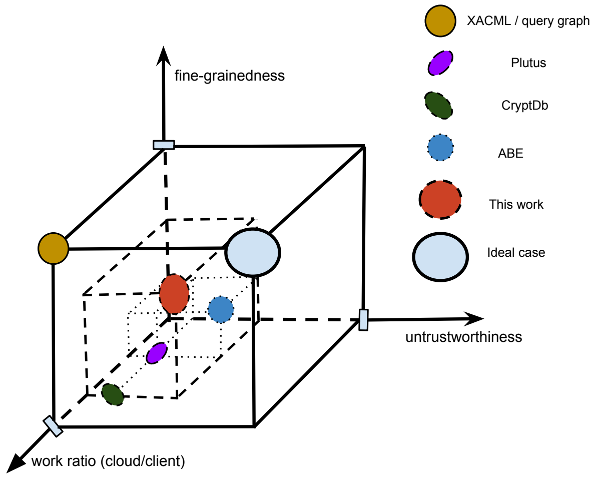


Figure 1: Design space for enforcing access control on the cloud

enterprise systems are thus migrating cloudward, for easy management and maintenance of their infrastructure [14]. At the same time, the easy and instant access to computing resource spawns a plethora small-to-medium size applications to be developed and deployed on the cloud [5]. The result of these trends is that a substantial and increasing amount of data is being hosted and maintained by one (or a small number of) cloud provider. On the one hand, such co-location of data makes it easy to share and perform analytics. On the other hand, the data owner may wish to prevent even its cloud service provider from having access to the data. Therefore, enforcing access control when data is outsourced to the cloud becomes more challenging.

Next we survey the current state-of-the-art in the field of access control on the cloud. We then present a system that achieves the above enumerated fine-grained access control while protecting data from a semi-honest cloud. The system outsources most of the heavy computation to the cloud, and makes use of a standard XML-based framework for high-level access control policy management. Finally, we discuss some open problems that need to be addressed before an *ideal* cloud-based access control system may be realized.

2 State of the Art

A way to characterize any system dealing with access control on a cloud environment would be along the three dimensions depicted in Figure 1. First, the *fine-grainedness* property indicates how many of the fine-grained policies are supported by the access control mechanism. For instance, if the system supports all four policies listed earlier, we can say that it achieves maximum fine-grainedness. On the other hand, if the system only supports an all-or-nothing policy (that means, an user either can access the whole data, or not at all), it is at the minimum with respect to this property. Second, the *untrustworthiness* property determines how untrustworthy the cloud is. If the cloud is completely trusted, this property is at its minimum, whereas if the cloud behaves in completely Byzantine manner, its untrustworthiness is at the maximal level. In between, the cloud may operate at different partially trusted settings. Finally, the *work ratio* property specifies how much work the cloud and the end-user client has to do, in order to carry out the system tasks in relation to each other. The client must be involved to some extent, but if it only has to do minimal, inexpensive work while the cloud carries out all the hard work, this property is at its maximum. In contrast, if the cloud performs very simple tasks such as storage or data distribution, the work ratio is near its minimum. The higher the value of this property is, the more beneficial it is to move to the cloud. In the following, we present the current state of the art, first distinguishing them along the untrustworthiness dimension then discussing how they differ from each other with respect to the other two properties. At this juncture, we

[5] `www.buddypoke.com`

will like to note that the dimensions are not necessarily completely ordered, nor are all relevant points of the dimensions necessarily known. For instance, one may identify other desirable fine-grained access control functionality not enumerated here. Furthermore, some of them may be more important than others. Likewise, different kinds of (mis)behaviors on the cloud service provider's part may have different adversarial effects, that may not all be ordered.

Trusted Cloud. If the cloud is trusted, it is equivalent to when the data owner runs the system on its private infrastructure. This eliminates the need for considering the work ratio property which is maximal because the client can outsource its entire operation to the cloud. The remaining concern is how to maximize the fine-grainedness property.

Traditional relational database systems enforce access control based on view, which essentially pre-computes additional data tables and sets all-or-nothing policies upon them. This approach is static: it requires the data owner to anticipates all possible access scenarios before creating views. For content-based policies such as filtering, where the attribute domain can be very large, this approach is clearly not scalable. Instead, a better practice is to generate views on-the-fly. It is particularly suitable for stream data which can be infinite in size. In particular, Carminati et al. [8] proposes an access control system for the Aurora data model [2] which defines policies as *query graphs* consisting of SQL-like operators that apply to data as it arrives. The authorized user receives the output of the query graph corresponding to the policy he is subject to. Dinh et al. [4] propose a similar approach, but focusing on extending the eXtendible Access Control Markup Language (XACML) framework for easy specification and enforcement of fine-grained policies. It exploits *obligation* elements of XACML, into which any user-defined function can be embedded and executed by the database server before returning data to the requesting user. Wang et al. [19] integrate the two approaches by providing a mechanism to translate the XACML policies into suitable query graphs.

Untrusted Cloud. In most cases, the data owner places great importance in maintaining the confidentiality of the data, either to protect its business asset, the personal or customer data, or to conform with the law. In these cases, it is essential to design safeguards under the assumption that the cloud is *untrusted*: for instance, it could deliberately behave maliciously or it may be infiltrated by a malicious party. A common adversary model for the cloud assumed in existing literature is a *semi-honest* model, i.e. the cloud is not entirely Byzantine.

Plutus [16] and CryptDB [18] are two cloud-based systems dealing with database outsourcing. They assume that the cloud is semi-honest with respect to data storage and retrieval. It means the cloud follows the protocol correctly but it tries to learn the raw data when storing and answering queries. Both systems employ encryption schemes, so that the cloud cannot see the plain-text data. Plutus focuses on revocation and efficiency, to this end it uses a broadcast encryption scheme. CryptDB supports more SQL-like operations such as search, aggregate and join, for which it uses a combination of advanced encryption schemes such as Paillier, order-preserving encryption and proxy re-encryption. These systems, however, support coarse-grained binary access control. In Plutus, the cloud's only job is to store and distribute ciphertexts, hence the work ratio falls heavily on the client side. CryptDb allows the cloud to compute certain operations on ciphertexts while answering user queries, therefore its work ratio is higher.

Attribute-Based Encryption (ABE) schemes [7, 12] allow more fine-grained access control to be enforced on cipher-text. An access tree T is defined over a set of attributes, such that the plaintext can be recovered only if T is evaluated to `true` over some given attributes. Two types of ABE exists: key-policy (KP-ABE) and ciphertext-policy (CP-ABE). They are interchangeable, but the former is more data-centric whereas the latter is user-centric. Essentially, ABE enables one to relax the semi-honest cloud model, i.e. the cloud may try to compromise access control: to grant data access to unauthorized users. ABE ensures that unauthorized access is not feasible nevertheless. ABE can readily be used to enforce filtering policy, therefore it achieves a higher fine-grainedness than CryptDb or Plutus does. Finally, the roles of the cloud in an ABE-based system such as [20] are mainly storage and distribution, therefore the work ratio is similar to that of [16] ([20] also focuses on access revocation by re-encrypting data).

Summary. Current state-of-the-art systems for cloud-based access control occupy a small zone in the design space, as illustrated in Figure 1, leaving much room for extension. The following section summarizes our proposal [3] that pushes the envelope further, albeit still remaining far from the ideal.

3 Outsourcing Fine-Grained Access Control — The pCloud approach

In the context of the pCloud[6] project, we have proposed an approach [3] that occupies an unique place in the design space (Figure 1). It assumes the same level of cloud trustworthiness as other ABE systems, while achieving higher level of fine-grainedness (as compared to both ABE and CryptDb) and better work ratio (as compared to both ABE and Plutus). The system supports both filtering and summary (sliding window) policies. The former is the result of using KP-ABE, while the latter is achieved by a combination of proxy re-encryption [13] and additive homomorphic encryption. The cloud transforms ABE ciphertexts into more simple Elgamal-like ciphertexts which are cheaper to decrypt. It also computes the sum over ciphertexts without compromising access control, i.e. the users authorized to access the sum only cannot learn the individual data. To achieve access control over sliding windows of size β, we use a set of blind factors $R = \{r_0, .., r_{\beta-1}\}$ when encrypting individual data, then give the authorized user the sum $\sigma = \sum_i r_i$. Using σ, the user can remove the blind factor in the sum of the data, but cannot learn individual r_i necessary for uncovering individual data. We assume that the data domain is small, so that discrete logarithm can be (pre-)computed for all of its values **(as an optimization)**.

KP-ABE: In the KP-ABE scheme, the mesage m is encrypted with a set of attributes A, and the user is given a policy P which is a predicate over a set of attributes $\{a_0, a_1, ..\}$. In addition, the user is given $S = \{s_{a_0}(y), s_{a_1}(y), ..\}$ created by the data owner using a secret sharing scheme such that y can be reconstructed if and only if $P(A) = \mathtt{true}$. The scheme guarantees that:

$$\mathtt{Dec}(\mathtt{Enc}(m, A), P, S) = m \leftrightarrow P(A) = \mathtt{true}$$

where $\mathtt{Enc}(.)$ and `Dec(.)` are the encryption and decryption function respectively.

Three-phase protocol for sliding window policy: During the ***setup phase***, the data owner creates a secret z, among other things [12]. Suppose the user is given a sliding window policy of size β starting from α. The data owner then computes:

$$\sigma(\alpha, \beta) = \sum_{j=0}^{\beta-1} 2^{\lceil(\alpha+j)/\beta\rceil} R[(\alpha + j)\ mod\ \beta]$$

Notice that for any $k > \alpha$ and $(k - \alpha)\ mod\ \beta = 0$, we have $\sum_{i=k}^{k+\beta-1} 2^{i+k/\beta}.R[(i + k)\ mod\ \beta] = 2^{\frac{k-\alpha}{\beta}}.\sigma(\alpha, \beta)$. Finally, it creates $S = \{s_{a_0}(y/z), s_{a_1}(y/z), ..\}$ for the policy $P := k \geq \alpha$. This way, the valid decryption $\mathtt{Dec}(.)$ will return $\phi(z, m)$ for a function ϕ such that m cannot be recovered without knowing z, and $\phi(z_1, m_1).\phi(z_2, m_2) = \phi(z_1.z_2, m_1.m_2)$. The tuple $(z, \sigma(\alpha, \beta), S)$ is sent to the user.

During the ***data outsourcing phase***, the data tuple (k, v_k) where $k = 0, 1, 2..$ is encrypted as:

$$c_k = \mathtt{Enc}(g^{v_k + 2^{\lceil k/\beta\rceil}.R[k\ mod\ \beta]}, \mu(k))$$

and sent to the cloud, where g is the generator of a multiplicative group and $\mu(k)$ maps k into a set of attributes.

[6] `http://sands.sce.ntu.edu.sg/pCloud/`

During the ***data streaming phase***, the cloud performs the transformation:

$$t_k = \texttt{Dec}(c_k, P, S) = \phi(z, g^{v+2^{\lceil k/\beta \rceil}.R[k \ mod \ \beta]})$$

For the i^{th} sliding window, the cloud computes $T_i = \prod_{j=0}^{\beta-1} t_{k+j} = \phi(z^\beta, g^{v_k+..+v_{k+\beta-1}+2^i.\sigma(\alpha,\beta)})$ for $k = \alpha + i.\beta$ and sends it to the user. Using z and $\sigma(\alpha, \beta)$, the user can recover $w_i = g^{v_k+..+v_{k+\beta-1}}$ from T_i. Finally, the average for window i^{th} is derived as $avg_i = \frac{\texttt{discreteLog}(w_i)}{\beta}$.

Protocol for filtering policy: The protocols for filtering policies are very similar to that for sliding window. The main differences are that the access policies may involve other conditions besides $\geq$, and that no blind factor is needed since authorized users can access individual data tuples.

Performance. We micro-benchmark our system for both sliding window and filtering policies.[7] The pairing implementation is of type A with 512-bit base field size [1]. The most expensive cryptographic operations are exponentiations and pairings, the latter of which are done by the cloud during transformation. The equality comparison in filtering policies results in 64 pairings, as opposed to the $k \geq \alpha$ conditions in sliding window policy requiring only 1 pairing for large values of k. The transformation takes maximum of $120ms$, while encryption at the data owner takes $179ms$.[8] These latencies are reasonable, because many data streams in practice generate data in intervals of seconds or minutes. Decryption time at the authorized users is around $0.3ms$ - an order of magnitude less than the transformation time at the cloud. This illustrates the benefit of having the cloud performing the heavy computation. The time taken for the setup phase, including pre-computing discrete logarithms for values in $[0, 5000]$ (which is an one-off cost) approximates $0.7s$.

XACML integration. Before starting the transformation operation, the cloud checks if the attributes associated with the ciphertext satisfy the access structure associated with the policy. We integrate XACML framework for systematic managing and matching of policies, so that redundant transformation at the cloud or decryption at the user can be avoided. In our system, the cloud runs an XACML instance, and each data stream is represented as a resource. The data owner defines XACML policies (filtering or summary), for each of which the cloud maintains a list of authorized users. The owner then specifies an XACML request for every ciphertext it sends to the cloud, which is then evaluated against the list of loaded policies. The result is a set of users to whom access to the data should be granted. Finally, the cloud performs transformation on the ciphertexts, or wait until collecting sufficient ciphertexts for the sliding window policies before transforming, and sends the new ciphertexts back to the user. In our micro-benchmark experiments, this additional layer of management incurs small overhead: the time taken for XACML processing with 1500 policies is under $5ms$.

4 Open Problems

Figure 1 hints at the open challenges. When the cloud is not trusted, it will be difficult if not infeasible to achieve the same level of work ratio as systems in the trusted settings can. In order to achieve any level of security with respect to the cloud, the data owner must perform some kind of data encryption. That a stronger level of security requires more expensive encryptions suggests that the more malicious the cloud is, the more work has to be done at the client. This must be taken into consideration when migrating to the cloud, as to balance the saving from outsourcing computation with the overheads of guaranteeing security.

A different level of fine-grainedness. Both filtering and summary policies (as supported in our work) involve simple computations before the data can be returned to the requesting user. These provide a simple data access abstraction, but higher-level abstractions involving more complex computations may be desirable. One example is a policy that grants access only to results of certain data mining algorithms or statistical

[7]The source code is available at `code.google.com/p/streamcloud`

[8]Experiments are run o desktop machines having 2 2.66Ghz DuoCore with 4GB of RAM

functions. When the cloud is trusted, this can be accomplished by extending XACML-based systems. In untrusted clouds, this must be accomplished using homomorphic encryptions. Although fully homomorphic encryption schemes are feasible in theory, making them practical for complex functions remain a challenge.

Decision support for determining sharing policies. So far, we have been assuming that the data owner knows what data is sensitive and what policies are to be chosen. In reality, these decisions are not obvious to make. For instance, many systems that collect user data and publish the anonymized versions have run into public relation disaster (Netflix and AOL, for example) because data can be linked to reveal sensitive information. Furthermore, the complexity and dynamics of social networks make it difficult for users of such systems to determine which policies to set for which friends [9]. *Differential privacy* can help reason about the former problem, as it provides a bound on how much privacy leakage would incur if the user decided to share something. A recent work [10] shows that the cloud can be delegated to ensure differential privacy on ciphertext, but it uses a encryption scheme which does not provide fine-grained access control. Likewise, decision support mechanisms to determine what to share with whom based on social and trust relations, as well as approaches to detect and prevent leakage of information are essential missing pieces.

Mitigating proactively malicious cloud. All systems covered in this article stop short of considering a fully malicious cloud. They assume the cloud adversary model to be semi-honest, i.e. it will execute the protocols truthfully while trying to compromise some security properties. Such an assumption does not suffice once the cloud has incentives to actively skip or subvert the delegated computation. Skipping computation may be driven by economic interest (the cloud doing less work while still charging the users), while competition may be a reason to distort computations. Both data owners and end users must be able to reliably detect such behaviors. This is a special case of verifiable computation, in which the client is able to verify the output of a function it outsourced to a third party. It has been shown that any computation can be outsourced with guaranteed input and output privacy [11], but the existing protocols are inefficient. A more practical approach may be probabilistic in nature. Other alternatives may include dividing the data or computation task over multiple cloud service providers, if the independence of these providers (alternatively, prevention of their collusion) can be guaranteed.

Other open problems. Even for the approach presented in Section 3, there are several unaddressed issues. We have not considered access revocation, which in our case requires the data owner to change the attribute set during encryption. We plan to investigate if existing revocable KP-ABE schemes [6] can be integrated, especially if they allow revocation to be outsourced to an untrusted cloud. Other interesting extensions are to add support for policies with negative attributes [17], and for encryption with hidden attributes. The former allows for a wider range of access policies, whereas the latter provides attribute privacy which is necessary when encryption attributes are the actual data.

5 Conclusions

In this article, we have discussed important challenges in designing a cloud-based fine-grained access control system. We have identified a core set of fine-grained access policies that are desirable in many real-life applications. Existing systems supporting this full set of policies assume a trusted cloud. For untrusted cloud, current state-of-the-art systems share similar semi-honest adversary models, while they differ in the levels of fine-grainedness in access control and the work ratio between the cloud and the client. We summarize our recent work that pushes the envelope [3] further. Finally, we outline a number of open problems that need to be overcome so that we can achieve, or at least get closer to the ideal cloud-based access control system.

References

[1] Key-policy attribute-based encryption scheme implementation. `http://www.cnsr.ictas.vt.edu/resources.html`.

[2] Daniel J. Abadi, Don Carney, Ugur Cetintemal, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael stone-

braker, Nesime Tatbul, and Stand Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–39, 2003.

[3] Dinh Tien Tuan Anh and Anwitaman Datta. Stream on the sky: Outsourcing access control enforcement for stream data to the cloud. *arXiv:1210.0660*, 2012.

[4] Dinh Tien Tuan Anh, Wang Wenqiang, and Anwitaman Datta. City on the sky: extending xacml for flexible, secure data sharing on the cloud. *Journal of Grid Computing*, pages 151–172, 2012.

[5] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: a stream data management benchmark. In *VLDB*, pages 480–91, 2004.

[6] Nuttapong Attrapadung. Revocation scheme for attribute-based encryption. RCIS Workshop, `http://www.rcis.aist.go.jp/files/events/2008/RCIS2008/RCIS2008_3-5_Nuts.pdf`, 2008.

[7] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–34, 2007.

[8] Barbara Carminati, Elena Ferrari, and Kian Lee Tan. Enforcing access control over data streams. In *SACMAT*, pages 21–30, 2007.

[9] Gorrell P. Cheek and Mohamed Shehab. Policy-by-example for online social networks. In *SACMAT*, pages 23–32, 2012.

[10] Ruichuan Chen, Alexey Reznichenko, and Paul Francis. Towards statistical queries over distributed private user data. In *NSDI*, 2012.

[11] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO'10*, August 2010.

[12] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS'06*, pages 89–98, 2006.

[13] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *20th Usenix conference on Security*, 2011.

[14] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *SIGCOMM*, 2010.

[15] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Gorackzko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: a distributed mobile sensor computing system. In *4th international conference on embedded networked sensor systems*, 2006.

[16] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: scalable secure file sharing on untrusted storage. In *FAST 2003*, 2003.

[17] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *CCS'07*, pages 195–203, 2007.

[18] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: a practical encrypted relational dbms. Technical Report MIT-CSAIL-TR-2011-005, CSAIL, MIT, 2011.

[19] Wen Qiang Wang, Dinh Tien Tuan Anh, Hock Beng Lim, and Anwitaman Datta. Cloud and the city: Facilitating flexible access control over data-streams. In *SDM*, 2012.

[20] Schucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable and fine-grained data access control in cloud computing. In *INFOCOM*, pages 534–542, 2010.

Policy Enforcement Framework for Cloud Data Management

Kevin W. Hamlen*, Lalana Kagal†, Murat Kantarcioglu*

*University of Texas at Dallas, †Massachusetts Institute of Technology

Abstract

Cloud computing is a major emerging technology that is significantly changing industrial computing paradigms and business practices. However, security and privacy concerns have arisen as obstacles to widespread adoption of clouds by users. While much cloud security research focuses on enforcing standard access control policies typical of centralized systems, such policies often prove inadequate for the highly distributed, heterogeneous, data-diverse, and dynamic computing environment of clouds. To adequately pave the way for robust, secure cloud computing, future cloud infrastructures must consider richer, semantics-aware policies; more flexible, distributed enforcement strategies; and feedback mechanisms that provide evidence of enforcement to the users whose data integrity and confidentiality is at stake. In this paper, we propose a framework that supports such policies, including rule- and context-based access control and privacy preservation, through the use of in-lined reference monitors and a trusted application programming interface that affords enforceable policy management over heterogeneous cloud data.

1 Introduction

Cloud computing security has rapidly emerged as a significant concern for businesses and end-users over the past few years. For example, in a 2010 survey, Fujitsu concluded that 88% of its customers have significant concerns about data integrity and privacy in the cloud [9]. While some cloud security issues are addressable via traditional techniques that have been used for decades to secure centralized, time-shared systems, others are endemic to the uniquely diverse and dynamic nature of cloud environments, and therefore demand new solutions [6].

Our prior research has identified at least three major categories of security challenges that are impeding information assurance in clouds: (1) semantic diversity of cloud data, (2) customer-cloud negotiated mobile computations, and (3) multi-party, cross-domain security, privacy and accountability policies.

Semantic diversity of data in clouds arises from the vast range of different datasets and data processing/querying tools that production-level clouds must process. These different datasets range from structured to unstructured data and data processing/querying frameworks range from MapReduce [7] (e.g., Hadoop [2]) to stream processing (e.g., [26]). This emerges as a security challenge because of the need to formulate policy languages that are sufficiently general to capture and relate permissible uses of security-relevant data with diverse semantics. For example, fine grained access control policies defined for streaming data applications could be vastly different from the applications that try to support SQL-like queries on relational data.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Mobile computations for clouds differ from traditional time-shared systems in that cloud infrastructures are seldom static or fully transparent to users. Further more, different cloud providers provide different internal network topology, storage model, job scheduling mechanisms, etc. These infrastructure choices often have security-relevant implications for users. For example, different versions of Hadoop may have different access control mechanisms that support different levels of granularity. In order to enforce their policies efficiently, users must be afforded a flexible range of enforcement options that allow them to choose the best enforcement strategy for each job given the (possibly evolving) constraints of the computing environment. Dually, the security adequacy of each option should be independently verifiable by the cloud provider in order to protect the cloud infrastructure and its other users.

Finally, the potential power of cloud computing stems in part from its ability to synergistically co-mingle large datasets from multiple organizations, and process distributed queries over the combined data for long periods of time. Policies for such an environment must be multi-party and history-based. For example, we may need to support contract-style data-sharing policies where organization X is willing to share data set D with organization Y if organization Y is willing to share data set D' with organization X.

In what follows, we describe how our prior work on a general cloud policy enforcement frameworks offers new, promising approaches for surmounting these challenges, and we recommend future work that will allow practical application of these technologies to clouds.

2 Proposed Approach

A general policy enforcement framework for cloud data management[1] must consider three important dimensions: (1) data type (e.g., relational data, RDF data, text data, etc.), (2) computation (e.g., SQL queries, SPARQL, Map/Reduce tasks, etc.), and (3) policy requirements (e.g., access control policies, data sharing policies, privacy policies, etc.) Given the wide range of available choices in each dimension, a policy enforcement framework must be highly flexible and must support different data processing requirements. To achieve these goals, we propose a policy-compliant data processing framework with the following modules:

- **Policy-reasoning module:** The main job of the policy-reasoning module is to map a policy to a specific set of tasks that will be executed on the data along with the submitted data processing task to enforce various policies. Based on the policy reasoning results, the policy reasoning module will output initial data pre-processing tasks, a modified data processing task, and the post-processing tasks for each submitted data processing task.

- **Data processing task rewriting module:** In some cases, using different pre-processing and post-processing tasks might result in different computational costs. In addition, to enable efficient computation, some of the pre-processing and post-processing tasks might need to be combined or simplified without provoking a policy violation. The data processing rewriting module can consider various rewriting strategies for enforcing policies.

- **Pre-processing module:** The pre-processing module executes the pre-processing tasks on the underlying data. For example, a pre-processing step might require the dataset to be anonymized with some privacy definition (e.g., l-diversity, t-closeness, k-anonymity, etc.) before any query is executed on it. In addition, pre-processing tasks could be used to filter sensitive data. For example, fine grained access control on relational data could be enforced by using a pre-processing task to create a view that can be given to each user-submitted SQL query.

- **Post-processing module:** In some cases, it might be more efficient to process the final results to enforce policies. For example, an accountability policy might require the creation of certain audit

[1]We here assume that the cloud system is trusted with enforcing given policies. Untrusted clouds are left to future work.

logs if the data processing task changed certain data records. Such policies could be enforced using post-processing tasks.

To efficiently implement the above modules, we need to able to specialize them for different data types, computation types and policy requirements. One way to achieve this goal is to create specialized modules for common data types, computations, and policies. For example, in our previous work [28], we built the above modules for enforcing role-based access control policies for a relational data store on the cloud that is processed using SQL queries. In essence, we have combined the Hadoop Distributed File System [27] with Hive [29] to provide a common storage area for storing large amounts of relational data and for running SQL like queries. Further, we have used an XACML [23] policy-based security mechanism to provide fine-grained access controls over the stored data. In this case, the policy reasoning module uses an XACML reasoning engine to check whether a user who submitted an SQL query has access to all the underlying table/views. The pre-processing module runs HiveQL queries to create materialized views that are accessed by user submitted queries. A query rewriting module can modify the submitted query based on the underlying view definitions for efficient query processing. In this case, post-processing may not be needed since pre-processing and/or query rewriting may be enough for enforcing basic access control policies.

Of course, such a specialized approached will not necessarily be applicable to other types of data, computations, and policies. For example, if the stored data is unstructured and computations executed on the data are arbitrary MapReduce jobs, then we need different policy enforcement techniques. In such scenarios, user submitted MapReduce jobs could be modified using in-lined reference monitoring techniques (see Section 2.2 for more details.).

2.1 Data-aware Policy Languages for Clouds

As mentioned previously, a policy enforcement framework for cloud must support a wide range of policies. In this section, we briefly summarize policies and policy languages previously proposed in the literature that are potentially enforceable by our framework.

Access controls are one of the most important policy classes that must be considered. At a general level, access control languages make assertions about users and their permissible operations [24]. Additionally, languages have been defined for making authorization decisions for policies that have been combined from various sources [3], as well as for supporting trust management (e.g., [32]). Furthermore, since the advent of XML and its acceptance as a *de facto* standard for information exchange, a number of access control languages based on XML have been proposed. An important example of XML-based access control is XACML[2], which provides an XML syntax for defining policies that constrain resources that may also be expressed in XML.

There have also been access control languages that are designed in a logic-based framework (e.g., [20]). The additional expressive power and formal, verifiable methodology offered by logic-based languages are particularly useful in the context of access control. Finally, access control languages have also been defined in the context of Semantic Web languages (e.g., Rei [16] and KAoS [30]). Semantic Web languages are based on description logics, which are a decidable subset of first order logic, and hence provide benefits that are similar to logic-based languages.

Group-Centric Secure Information Sharing (g-SIS) (e.g., [18]) is an example of a family of access control models that is tailored to suit the requirements of information sharing on the cloud. The family of models in g-SIS is based on the notion of brining users and resources together into a group for purposes of information sharing. In other words, this means that users and resources must be present in the system simultaneously for the users to be able to access the resources. In addition, the family of g-SIS models is based on the following two principles:

[2]http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html

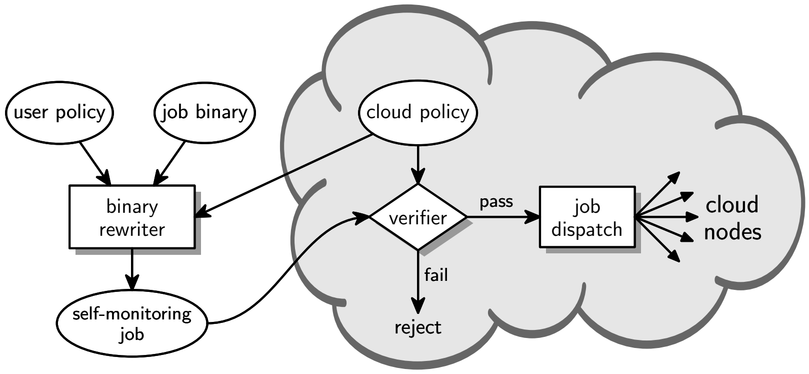


Figure 1: A cloud framework with policy-enforcement based on certifying IRMs

- **Share but differentiate:** The sharing aspect is achieved through users joining a group and then adding information to that group. However, the access that a user is granted to resources is based on several factors—for example, the time at which a user is added to the group relative to the time at which the information was posted to the group, and other user-configurable attributes.

- **Multiple groups:** This principle refers to the notion that various types of relationships can hold between different groups (e.g., hierarchical, mutual exclusion, etc.) with a possibly overlapping set of users.

Research devoted to the g-SIS family of models has included the development of a formal model for g-SIS using linear temporal logic (LTL) [21], the specification of core properties that g-SIS models must satisfy, as well as extensions that show how g-SIS models allow secure and agile information sharing when compared with traditional access control techniques, and finally, the development of a "stateful" enforceable specification of g-SIS [19].

An alternative approach to provenance-aware access control is to tailor the language to suit the requirements (e.g., policy enforcement based on an aggregation of applicable policies, redaction policies, etc.) [22]. This proposed language uses an XML syntax and specifies a variety of tags (viz., target, effect, condition, and obligations) to capture various access control use-cases that arise in the domain of provenance. However, the language is not able to capture resources with arbitrary path lengths that occur within a provenance graph. Therefore, a resource to be protected must be identified *a priori*, rather than being passed as a parameter at runtime. The task of identifying resources *a priori* might be infeasible, since there might be an exponential number of resources in a provenance graph.

Subsequent work has addresses this path-length drawback through the use of regular expressions to define resources requiring protection [4]. The use of regular expressions allows resources with arbitrary path lengths to be defined and used at runtime rather than having to create resources *a priori*. The same authors have also extended the notion of redaction to provenance graphs [5]. They use a graph grammar approach [8] that rewrites a high-level specification of a redaction policy into a graph grammar rule that transforms the original provenance graph into a redacted provenance graph.

2.2 Flexible Cloud Policy Enforcement

One way to provide cloud customers maximum flexibility with regard to policy enforcement is to permit the enforcement mechanism to reside within the jobs, with suitable checking on the cloud side to ensure that the job's self-enforcement is adequate. Such a cloud framework is illustrated in Figure 1. For example, a job expressed as a Java bytecode binary (as in typical Hadoop MapReduce clouds) can self-enforce an access control policy by voluntarily restricting its accesses to policy-permitted resources. If the full policy is not known at code-generation time, the cloud can even provide a trusted system API that job code may consult at runtime to discover policies to which it is subject and self-censor its resource accesses accordingly.

As a simple illustration of how such self-enforcement can be more efficient than cloud-implemented, system-level enforcement, consider a job J that counts database records $r \in D$ satisfying some predicate $P_J(r)$, and consider a policy C that prohibits J from accessing records that falsify a policy-prescribed predicate $P_C(r)$. To enforce this policy, a system-level implementation might intercept all attempts by J to access records r, denying those for which $P_C(r)$ is falsified. Alternatively, job J could self-enforce this policy by implementing $P_C(r)$ within its own code, but only evaluating it on records r that have already satisfied $P_J(r)$. In both cases, job J satisfies the policy and counts the set of records that satisfy conjunction $P_C(r) \wedge P_J(r)$. However, if P_C is more computationally expensive than P_J and few records satisfy P_J, the self-enforcement approach could be far more efficient.

Self-enforcement need not impose an implementation burden on clients if the inclusion of the enforcement mechanism into the job code can be automated. Our prior work has demonstrated that enforcement of safety policies—including access control policies—can be in-lined into arbitrary, untrusted, Java binaries fully automatically through the use of *aspect-oriented in-lined reference monitors* (IRMs) [10, 14, 15]. IRMs in-line the logic of traditionally system-level reference monitors into untrusted code to produce policy-compliant, self-monitoring code [25]. The in-lining process identifies potentially security-relevant instructions in the untrusted code and surrounds them with guard code that preemptively detects impending policy violations at runtime. When an impending violation is detected, the guard code intervenes by halting the job prematurely or taking some other corrective action (e.g., throwing a catchable exception or rolling back to a consistent state).

For example, a simple binary rewriter might replace all instructions $\mathit{read}()$, which read a new database record, with a wrapper function of the form

$$\big(\textbf{let}\ r = \mathit{read}()\ \textbf{in}\ (\textbf{if}\ P_C(r)\ \textbf{then}\ r\ \textbf{else}\ \mathit{error})\big) \tag{2}$$

The result of such a transformation is code that self-enforces policy C. More sophisticated rewriters can in-line such guard code more intelligently, such as by shifting the check to time-of-use sites instead of time-of-read sites when doing so improves performance, or by distributing the implementation of P_C across multiple nodes when P_C is computationally expensive (e.g., when r is large).

IRM implementations and the policies they enforce can be elegantly expressed using *aspect-oriented programming* (AOP) [31]. AOP has been used for over a decade to implement cross-cutting concerns in large source codes, and there is a rich family of production-level compilers and programming languages that support it. It extends traditional object-oriented programming with *aspects*, which consist of *pointcuts* and *advice*. Pointcuts are similar to regular expressions, but match sets of program operations instead of strings. The compiler or aspect-weaver for an AOP system in-lines the aspect-supplied advice code into the target program at every code point that matches the pointcut. In the context of IRMs, pointcuts can be leveraged to specify policy-relevant program operations (e.g., *read*), and advice can be leveraged to specify guard code for those operations (e.g., the computation in 2).

Thus, AOP and AOP-based specification languages constitute a powerful and well-developed paradigm for enforcing a broad class of security policies as IRMs. In such a framework, users and clouds specify policies using a declarative policy language, such as SPoX [10], from which an automated rewriter synthesizes appropriate aspects and weaves them into the target code at the binary level. The result is binary code that transparently self-enforces the policies without any manual intervention from the user.

The use of IRMs as a basis for enforcing policies mandated by the cloud or by other clients (e.g., owners of shared data accessed by untrusted, third-party jobs) is only feasible if the cloud can independently verify that submitted jobs satisfy the policies to which they are subject. While such code properties are in general undecidable [13], recently a series of technologies has emerged that permit a broad class of powerful IRM implementations to be verified fully automatically via type-checking [12], contract-based certification [1], or model-checking [11]. By implementing one of these algorithms, a trusted cloud can safely permit jobs to self-enforce mandatory access control policies, yet statically verify that this self-enforcement is sound prior to the job's deployment.

The cloud framework depicted in Figure 1 combines these ideas to implement a flexible policy enforcement strategy based on certifying IRMs. Jobs expressed as binary code are rewritten automatically on the client side in accordance with both user-specified (i.e., discretionary) and cloud-specified (i.e., mandatory) policies. The result of the rewriting is a new binary that self-enforces the desired policies. When the job is submitted to the cloud, the cloud first verifies that the submitted code has been instrumented with security checks sufficient to enforce the mandatory policies. Once it passes verification, the job can then be safely dispatched to the rest of the cloud without the need for additional system-level monitoring.

3 Conclusion

In this paper, we outlined a general policy enforcement framework needed for policy-compliant cloud data management. We discussed different policy types applicable for cloud data management ranging from data sharing policies to traditional access control policies, and showed how various techniques such as IRMs could be used to enforce such policies in a flexible, user-driven, but cloud-certified manner. Our proposals assumed that the cloud infrastructure is trusted to enforce or certify the enforcement of the specified policies. In our future work, we plan to explore how such policies could be enforced on semi-trusted and/or untrusted cloud infrastructures (cf., [17]).

References

[1] I. Aktug, M. Dam, and D. Gurov. Provably correct runtime monitoring. In J. Cuellar, T. Maibaum, and K. Sere, editors, *Proceedings of the 15th International Symposium on Formal Methods (FM)*, pages 262–277, 2008.

[2] Apache™ Hadoop®. http://hadoop.apache.org.

[3] P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and Systems Security (TISSEC)*, 5(1):1–35, 2002.

[4] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. A language for provenance access control. In *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 133–144, 2011.

[5] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 93–102, 2011.

[6] Y. Chen, V. Paxson, and R. H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EE & CS Dept., U.C. Berkeley, 2010.

[7] J. Dean and S. Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[8] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, Berlin, 2006.

[9] Fujitsu. Personal data in the cloud: A global survey of consumer attitudes. Technical report, Fujitsu Research Institute, 2010.

[10] K. W. Hamlen and M. Jones. Aspect-oriented in-lined reference monitors. In Ú. Erlingsson and M. Pistoia, editors, *Proceedings of the 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 11–20, 2008.

[11] K. W. Hamlen, M. M. Jones, and M. Sridhar. Aspect-oriented runtime monitor certification. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 126–140, 2012.

[12] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Certified in-lined reference monitoring on .NET. In V. C. Sreedhar and S. Zdancewic, editors, *Proceedings of the 1st ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 7–16, 2006.

[13] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(1):175–205, 2006.

[14] M. Jones and K. W. Hamlen. Enforcing IRM security policies: Two case studies. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 214–216, 2009.

[15] M. Jones and K. W. Hamlen. Disambiguating aspect-oriented security policies. In J.-M. Jézéquel and M. Südholt, editors, *Proceedings of the 9th International Conference on Aspect-Oriented Software Development (AOSD)*, pages 193–204, 2010.

[16] L. Kagal, T. W. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 63–74, 2003.

[17] S. M. Khan and K. W. Hamlen. Hatman: Intra-cloud trust management for Hadoop. In *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 494–501, June 2012.

[18] R. Krishnan and R. S. Sandhu. A hybrid enforcement model for group-centric secure information sharing. In *Proceedings of the International Conference on Computational Science and Engineering (CSE)*, volume 3, pages 189–194, 2009.

[19] R. Krishnan and R. S. Sandhu. Authorization policy specification and enforcement for group-centric secure information sharing. In *Proceedings of the 7th International Conference on Information Systems Security (ICISS)*, pages 102–115, 2011.

[20] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 114–130, 2002.

[21] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[22] Q. Ni, S. Xu, E. Bertino, R. S. Sandhu, and W. Han. An access control language for a general provenance model. In *Proceedings of the 6th VLDB Workshop on Secure Data Management (SDM)*, pages 68–88, 2009.

[23] OASIS XACML Technical Committee (B. Parducci and H. Lockhart, chairs; E. Rissanen, editor). eXtensible Access Control Markup Language (XACML) version 3.0. Oasis, 2010.

[24] P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In *Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design: Tutorial Lectures*, pages 137–196, 2000.

[25] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[26] Storm: Distributed and fault-tolerant realtime computation. http://storm-project.net.

[27] K. Svachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[28] B. M. Thuraisingham, V. Khadilkar, A. Gupta, M. Kantarcioglu, and L. Khan. Secure data storage and retrieval in the cloud. In *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–8, 2010.

[29] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. In *Proceedings of the VLDB Endowment*, volume 2(2), pages 1626–1629, 2009.

[30] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference*, pages 419–437, 2003.

[31] M. Wand, G. Kiczales, and C. Dutchyn. A semantics for advice and dynamic join points in aspect-oriented programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(5):890–910, 2004.

[32] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and Systems Security (TISSEC)*, 6(1):1–42, 2003.

Secure Data Processing over Hybrid Clouds

Vaibhav Khadilkar #1, Kerim Yasin Oktay *1, Murat Kantarcioglu #2 and Sharad Mehrotra *2
#The University of Texas at Dallas
1vvk072000@utdallas.edu, 2muratk@utdallas.edu
*University of California, Irvine
1koktay@uci.edu, 2sharad@ics.uci.edu

Abstract

A hybrid cloud is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability [1]. The emergence of the hybrid cloud paradigm has allowed end-users to seamlessly integrate their in-house computing resources with public cloud services and construct potent, secure and economical data processing solutions. An end-user may be required to consider a variety of factors, which include several hybrid cloud deployment models and numerous application design criteria, during the development of their hybrid cloud solutions. Although a multitude of applications could be developed through a combination of the aforementioned deployment models and design criteria, the common denominator among these applications is that they partition an application's workload over a hybrid cloud. Currently, there does not exist a framework that can model this workload partitioning problem such that all the previously mentioned factors are considered. Therefore, in this paper we present our vision for the formalization of the workload partitioning problem such that an end-user's requirements of performance, data security and monetary costs are satisfied. Furthermore, to demonstrate the flexibility of our formalization, we show how existing systems such as [2] [3] can be derived from our general workload partitioning framework through an instantiation of the appropriate criteria.

1 Introduction

The emergence of cloud computing has created a paradigm shift within the IT industry by providing users with access to high quality software services (SaaS), robust application development platforms (PaaS) and sophisticated computing infrastructures (IaaS). Furthermore, the utilization of a pay-as-you-use pricing model for usage of cloud services is a particularly inviting feature for users, since it allows them to significantly lower their initial investment cost towards acquiring a cloud infrastructure. A hybrid cloud is a particular cloud deployment model that is composed of two or more distinct cloud infrastructures (private, community, or public) that remain autonomous entities, but are interleaved through standardized or proprietary technology that enables data and application portability [1]. A growing number of organizations have turned to such a **hybrid cloud model** [4] [5], which allows them to seamlessly integrate their private cloud infrastructures with public cloud service providers. A hybrid cloud model enables users to process organization-critical tasks on their private infrastructure while allowing repetitive, computationally intensive tasks to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

be outsourced to a public cloud. Moreover, adopting a hybrid cloud model increases throughput, reduces operational costs and provides a high level of data security.

There are several flavors of hybrid cloud deployment models that are available to a user. One of the choices is the **Outsourcing model**, in which users outsource all tasks to public cloud service providers. Additionally, a user's private cloud infrastructures are mainly used to perform post-processing operations such as filtering incorrect results or decrypting encrypted results. Some features of the Outsourcing model include ease of application development and deployment, reduction in financial expenditure and guaranteed levels of performance through the use of Service Level Agreements (SLA's). A second choice is the **Cloudbursting model**, in which users primarily use an in-house cloud infrastructure for deploying applications and use public cloud services to mitigate sudden bursts of activity associated with an application. A Cloudbursting model provides users with several advantages such as adaptability to changing computational capacity requirements and cost savings through efficient use of public cloud resources. A third choice is the **Hybrid model**, in which users could host applications that operate over sensitive information on a private infrastructure while outsourcing less critical applications to a public cloud service provider. Such a deployment model offers a higher level of throughput, an enhanced data security and an overall reduction in financial costs. Given the various choices we have just described, a user needs to make a mindful selection of a particular deployment model based on their application requirements.

In addition to the deployment models presented above, a user also needs to consider a variety of criteria for designing a hybrid cloud application. The single most important criterion is **Performance**, since any design solution must strictly adhere to a user's performance requirements. The performance of a hybrid cloud application depends on several criteria such as the data model used to capture information and the data representations used to store data on a public cloud. The second important criterion that merits a user's consideration is **Data Disclosure Risk**, since a hybrid cloud application outsources tasks, and implicitly data, to a public cloud, thereby creating a potential risk if the data is leaked. The data disclosure risk is dependent on factors such as the representation used to store data on a public cloud and whether a selected representation discloses information during data processing. The third important criterion that deserves a user's attention is **Resource Allocation Cost**, since an application's usage of cloud services leads to expenses that must be covered by an organizational budget. The resource allocation cost is contingent on the cloud vendor and type of services being commissioned. The last essential touchstone that a user should consider is **Private Cloud Load**, since for certain deployment models, namely Outsourcing and Cloudbursting, a user would necessarily want to limit the amount of processing that is performed on a private cloud. In practice, the load generated on a private cloud primarily depends on the model used to capture and process data. Given the multitude of design criteria we described, a user is required to make selections in a way that effectively addresses their performance, security and financial requirements.

In this paper, we begin by identifying the most notable criteria, which were briefly outlined earlier, that drive the design of an effective hybrid cloud solution. In addition, we also tabulate the applicability of these criteria to various cloud deployment models, which were introduced earlier. An observation to be made at this point is that, although a user is required to consider a variety of factors, such as several hybrid cloud deployment models as well as numerous application design criteria, the common denominator among any applications that are developed using the aforementioned factors is that they partition an application's workload over a hybrid cloud. In this paper, we formalize this workload partitioning problem as a mechanism for maximizing a workload's performance and we subsequently develop a framework for distributing an application's workload over a hybrid cloud such that an end-user's requirements with respect to performance, data disclosure risk, resource allocation cost and private cloud load are satisfied. We then describe how existing systems such as [2] [3] can be derived from the general workload partitioning framework through an instantiation of the appropriate parameters.

Our primary technical contributions are listed below:

- We identify the most significant criteria that drive the design of an effective hybrid cloud solution. In addition, we demonstrate the applicability of these criteria to various cloud deployment models.
- We formalize the workload partitioning problem as a mechanism for maximizing workload performance. Our formalization allows us to plug in various models for metrics that have the greatest impact towards the effectiveness of a hybrid cloud deployment model. In addition, our formalization allows an end-user to experiment with different levels of restrictions for public cloud usage, until they achieve the right mix of performance, security and financial costs.
- We demonstrate the flexibility of our formalization by showing how existing systems such as Sedic [2] as well as the work given in [3], henceforth referred to as Hybrid-I, can be derived from the general workload partitioning framework through an instantiation of the appropriate design criteria.

The rest of the paper is organized as follows: In Section 2 we present several key design criteria that we believe are essential towards the development of an effective hybrid cloud solution. Then, Section 3 presents a general formalization of the workload partitioning problem that is applicable to any hybrid cloud deployment model using the design criteria outlined in Section 2. After that, Section 4 describes how existing systems can be derived from the general workload partitioning framework based on a specification of concrete values for the appropriate design criteria. Finally, we describe our conclusions and future work in Section 5.

2 Design Criteria for Hybrid Cloud Models

In this section, we present a brief overview of the design criteria that provide the greatest contribution towards an effective hybrid cloud solution. Furthermore, Table 2 shows how these criteria are applicable to hybrid cloud models (*viz.* Outsourcing, Cloudbursting and Hybrid) as well as a Private-only cloud.

Performance: This criterion is the single most important one for the adoption of hybrid clouds, since a user would be willing to consider a cloud approach only if it meets their evolving performance requirements. In the context of hybrid clouds, there are several, mutually conflicting metrics that could be used to measure performance. These include, query response time and network throughput, among others. The performance of a hybrid cloud model is in turn dependent on several factors such as data model, sensitivity model, *etc*.

Data Disclosure Risk: This factor estimates the risk of disclosing sensitive data to a public cloud service provider, albeit in an appropriately encrypted form [6]. The risk is contingent on the the sensitivity and security models defined by a user. Furthermore, the risk could be measured using a simple metric such as the number of sensitive cells exposed to a public cloud [3] or a more complex analytical [7] or entropy-based [8] technique.

Resource Allocation Cost: This criterion measures the financial cost (in terms of $) engendered by the incorporation of some type of public cloud services into hybrid cloud models. The cost can be classified into the following two broad categories: (i) On-premise: This category measures the cost incurred in acquiring and maintaining a private cloud. (ii) Cloud: This category can be further sub-divided as follows: (a) Elastic: A user is charged only for the services they use (pay-as-you-use). (ii) Subscription: A user is charged a decided fee on a regular basis (fixed). The financial cost of an end-user's hybrid cloud model implementation is dependent on several factors such as the data model/query language, storage representation, *etc*.

Private Cloud Load: This touchstone estimates the load on a private cloud generated as a result of processing some part of a user's workload. This criterion is particularly appropriate in the context of the Outsourcing and Cloudbursting deployment models, where the goal of a user is to avoid processing any data or processing a small amount of data on a private cloud. The load on a private cloud could be measured using a variety of metrics such as workload response time, total number of I/O operations performed when a workload is processed, *etc*.

Observations: There are several observations to be made from the criteria we have listed above.

Table 2: Design Criteria and their Applicability to Cloud Models

Design Criteria	Private-only	Outsourcing	Cloudbursting	Hybrid
Performance	✓	✓	✓	✓
Data Disclosure Risk	×	✓	✓	✓
Resource Allocation Cost	On-premise	Cloud	Both	Both
Private Cloud Load	×	✓	✓	✓

- The different criteria are tightly coupled with one another, thus requiring a methodical selection process to successfully accomplish an end-user's requirements.
- The main distinguishing characteristic between the Outsourcing model *vs.* the Cloudbursting and Hybrid models is the Data Disclosure Risk. In the Outsourcing model, the disclosure risk is higher than the Cloudbursting and Hybrid models, since the entire dataset and workload are outsourced to a public cloud[1]. On the other hand, the disclosure risk in the Cloudbursting and Hybrid models can be configured as an adjustable parameter, thus causing the overall risk in these models to be lower than the Outsourcing model.
- Although the Cloudbursting and Hybrid models appear to overlap in terms of the criteria described above, there are two important differences between them: (i) In the Cloudbursting model, private cloud data is always replicated on a public cloud. The level of replication, *viz*, partial or full, is dependent on an end-user's choice. However, in the Hybrid model, an end-user decides whether data replication is performed. (ii) In the Cloudbursting model, computations are pushed to a public cloud only when the generated load begins to exhaust private cloud resources. In the Hybrid model, a user's preference dictates whether private cloud load is used as a criterion for distributing a workload.

3 Workload Partitioning Problem

In this section, we formalize the workload partitioning problem, WPP, for a hybrid cloud setting, using the design criteria we outlined in Section 2. The goal of WPP is to distribute a workload W, and implicitly a dataset R, over a hybrid cloud deployment model such that the overall performance of W is maximized. Additionally, the problem specification is bounded by the following constraints: (i) **Data Disclosure Risk**: The risk an end-user is willing to accept due to disclosure of sensitive data stored on a public cloud. (ii) **Public Cloud Resource Allocation Cost**: A user-defined upper bound on monetary costs, which limits the amount of public cloud services that could be leased for processing data. (iii) **Private Cloud Load**: The permissible capacity to which private cloud resources could be commissioned for processing data.

WPP **Definition**: Given a dataset R and workload W, WPP can be modeled as an optimization problem whose goal is to find a subset $W_{pub} \subseteq W$ of the workload, and implicitly a subset $R_{pub} \subseteq R$ of the dataset such that the overall performance of W is maximized.

$$\begin{array}{ll} \text{maximize} & Performance(W, W_{pub}) \\ \text{subject to} & (1)\ Risk(R_{pub}, Rep) \leq DISC\_CONST \\ & (2)\ Pricing(R_{pub}, W_{pub}) \leq PRA\_CONST \\ & (3)\ Load(W - W_{pub}) \leq LOAD\_CONST \end{array}$$

[1]Public cloud services such as Amazon S3 allow users to store data in an encrypted format at no additional monetary costs [9]. This facility ensures that data is protected when it is unused, however, the data is in cleartext form when it is brought into memory during processing, and hence is susceptible to memory attacks at this time [10].

where $DISC\_CONST$, PRA_CONST and $LOAD\_CONST$ denote the maximum admissible data disclosure risk, public cloud resource allocation cost and private cloud load as specified by an end-user. The general formalization of WPP given above extracts and presents the essential components of the workload partitioning problem in the context of various hybrid cloud deployment models. Furthermore, such a general framework allows us to construct several practical hybrid clouds by instantiating each of the criteria specified in Section 2 with different values. Additionally, a general specification enables us to systematically analyze the interdependence between the design criteria and thus assist users in making informed choices for the various criteria.

The general formalization of WPP includes a high-level mathematical definition of various metrics, namely performance, data disclosure risk, public cloud resource allocation cost and private cloud load, which collectively assist us in measuring the effectiveness of a hybrid cloud deployment model. This high-level definition needs to be further refined for a particular hybrid cloud variant based on the values specified for the various design criteria outlined earlier. Therefore, in the subsequent section, we present specific instantiations of the applicable metrics, as defined by Sedic and Hybrid-I.

4 Sample Variants of WPP for the Hybrid Cloud Deployment Model

In this section, we demonstrate the flexibility of our formalization by showing how existing systems such as Sedic [2] and Hybrid-I [3] can be derived from the general workload partitioning framework through a specification of concrete values for the appropriate design criteria we identified earlier.

Sedic: An inherent drawback to existing cloud computing frameworks, such as MapReduce, is their inability to automatically partition a computational task such that computations over sensitive data are performed on an organization's private cloud, while the remaining data is processed on a public cloud. The goal of Sedic is to address this drawback by enhancing the MapReduce framework with special features that allow it to partition and schedule a task over a hybrid cloud according to the security levels of the data used by the task.

The workload partitioning problem definition for Sedic [2] can be constructed by using the values given in Table 3 for the various design criteria. Note that, Sedic also uses the following specifications: (i) Data Model: Key-Value. (ii) Data Partitioning Model: None. (iii) Data Replication Model: Full replication of non-sensitive data to public cloud. (iv) Sensitivity Model: Sensitivity is defined at data-level using a labeling tool. (v) Security Model for Public Clouds: All sensitive data is sanitized to 0. (vi) Workload Model: A single MapReduce job.

Table 3: Design Criteria Specification for Sedic

Design Criteria	Specification
Performance	Overall Task Execution Time
Data Disclosure Risk	0, *viz.*, no sensitive data is exposed
Resource Allocation Cost	None
Private Cloud Load	Not considered

WPP **Definition for Sedic**: Since Sedic supports single MapReduce jobs, W can be modeled as a workload of tasks T, where a task is either a Map or Reduce task. Then, WPP can be defined as follows for Sedic: Given a dataset R and a task workload T, a variant of WPP for Sedic can be modeled as an optimization problem whose goal is to find subsets $T_{pub} \subseteq T$ and $R_{pub} \subseteq R$ such that the overall execution time of T is minimized.

$$\begin{aligned} \text{minimize} \quad & Performance(T, T_{pub}) \\ \text{subject to} \quad & (1)\ Risk(R_{pub}, Rep) \leq DISC\_CONST \end{aligned}$$

where, as before, $DISC\_CONST$ denotes the maximum permissible data disclosure risk, which is 0 for Sedic, since no sensitive information can be leaked to a public cloud. In addition, the following observations can be made from the WPP definition of Sedic based on the specifications given above: (i) A data item $R_i \in R$ denotes either a Key or Value, since Sedic uses the Key-Value data model, no partitioning and a full data replication model. (ii) The set Rep consists of two representations, namely "plaintext" and "0", since Sedic sanitizes all sensitive data stored on a public cloud to 0.

We now provide specific instantiations of performance and data disclosure risk that suitably capture aspects of the metrics that are relevant to the problem domain modeled by Sedic.

Performance: As stated earlier, Sedic uses the *overall task execution time* of workload T, denoted as $ORunT(T, T_{pub})$, as an indicator of performance. Consequently, the objective function of WPP aims to minimize the overall execution time of a given task workload T. The execution time of tasks in T over a hybrid cloud, given that tasks in T_{pub} are executed on a public cloud can be represented as follows:

$$Performance(T, T_{pub}) = ORunT(T, T_{pub}) = \max \begin{cases} \sum\limits_{t \in T_{pub}} runT_{pub}(t) \\ \sum\limits_{t \in T - T_{pub}} runT_{priv}(t) \end{cases}$$

Note that $T_{pub} \subseteq T$, otherwise it is undefined. Additionally, $runT_x(t)$ denotes the estimated running time of task $t \in T$ at site x where x is either a public ($x = pub$) or private ($x = priv$) cloud. In practice, a methodology such as that given in [11] can be used to estimate the running time of a task t as follows:

$$runT_x(t) = \begin{cases} totalMapTime = \begin{cases} cReadPhaseTime + cMapPhaseTime + cWritePhaseTime, & \text{if } pNumReducers = 0 \\ cReadPhaseTime + cMapPhaseTime + cCollectPhaseTime + \\ cSpillPhaseTime + cMergePhaseTime, & \text{if } pNumReducers > 0 \end{cases} \\ totalReduceTime = cShufflePhaseTime + cMergePhaseTime + cReducePhaseTime + cWritePhaseTime \end{cases}$$

where the semantics associated with the different variables used above are given in Table 4. An interested reader can refer to the technical report given in [11] for additional details.

Data Disclosure Risk: Sedic uses a data labeling tool to mark sensitive subsets, R_i, of a dataset R. Furthermore, Sedic sanitizes any marked out sensitive data, which needs to be stored on a public cloud, to 0. A combination of these two factors ensures that no sensitive data is exposed to a public cloud, *viz.* $Risk(R_{pub}, Rep) = 0$. Moreover, no sensitive information is leaked from a public cloud, since all sensitive values are sanitized to the same value, *viz.* 0. Finally, since no sensitive data is exposed to a public cloud, Sedic ensures that the data disclosure risk constraint, namely $DISC\_CONST$, which has a value of 0 for Sedic, is not violated.

Hybrid-I: A common characteristic across all hybrid cloud applications is that they partition the application's computational workload, and implicitly the data, over a hybrid cloud. However, a user has a multitude of computation partitioning choices based on their desired application requirements. Moreover, it is infeasible to construct applications over each of the possible computation partitioning choices. The goal of Hybrid-I is to formalize the computation partitioning problem over hybrid clouds such that an end-user's desired requirements are achieved. Additionally, Hybrid-I provides a dynamic programming solution to the computation partitioning problem, when the underlying workload consists of Hive queries and the dataset is assumed to be relational.

The workload partitioning problem definition for Hybrid-I can be constructed through an instantiation of the various design criteria using the values given in Table 5. Note that, Hybrid-I also uses the following

Table 4: Semantics of Variables used in the estimation of the running time of a task t

Variable	Semantics
$cReadPhaseTime$	The time to perform the Read phase in a Map task
$cMapPhaseTime$	The time to perform the Map phase in a Map task
$cCollectPhaseTime$	The time to perform the Collect phase in a Map task
$cSpillPhaseTime$	The time to perform the Spill phase in a Map task
$cMergePhaseTime$	The time to perform the Merge phase in a Map/Reduce task
$cShufflePhaseTime$	The time to perform the Shuffle phase in a Reduce task
$cReducePhaseTime$	The time to perform the Reduce phase in a Reduce task
$cWritePhaseTime$	The time to perform the Write phase in a Map/Reduce task
$totalMapTime$	The overall time to perform a Map task
$totalReduceTime$	The overall time to perform a Reduce task

specifications: (i) Data Model: Relational. (ii) Data Partitioning Model: Vertical. (iii) Data Replication Model: Partial replication of data to public cloud. (iv) Sensitivity Model: Attribute-level. (v) Security Model for Public Clouds: Bucketization [12]. (vi) Workload Model: Hive[2] queries in batch form.

Table 5: Design Criteria Specification for Hybrid-I

Design Criteria	Specification
Performance	Overall Query Execution Time
Data Disclosure Risk	No. of sensitive tuples exposed to Public cloud
Resource Allocation Cost	Cloud - Elastic
Private Cloud Load	Not considered

WPP **Definition for Hybrid-I**: Since Hybrid-I uses Hive queries in batch form, the workload W can be modeled as a set of Hive queries Q. Then, the WPP definition for Hybrid-I can be given as follows: Given a dataset R and a query workload Q, a variant of WPP for Hybrid-I can be modeled as an optimization problem whose goal is to find subsets $Q_{pub} \subseteq Q$ and $R_{pub} \subseteq R$ such that the overall execution time of Q is minimized.

$$\begin{aligned} \text{minimize} \quad & Performance(Q, Q_{pub}) \\ \text{subject to} \quad & (1)\ Risk(R_{pub}, Rep) \leq DISC\_CONST \\ & (2)\ Pricing(R_{pub}, Q_{pub}) \leq PRA\_CONST \end{aligned}$$

where, as before, $DISC\_CONST$ and PRA_CONST denote the maximum permissible data disclosure risk and public cloud resource allocation cost. In addition, the following observations can be made from the WPP definition of Hybrid-I based on the specifications given above: (i) A data item $R_i \in R$ denotes an attribute of a relation of R, since Hybrid-I uses the relational data model, a vertical data partitioning model and a partial data replication model. (ii) Rep consists of two representations, namely "plaintext" and "bucketization", since Hybrid-I uses a column-level sensitivity model along with bucketization as the security model for public clouds.

[2] http://hive.apache.org/

Again, we provide specifications of performance, data disclosure risk and resource allocation cost that aptly capture aspects of the metrics that are relevant to the problem domain modeled by Hybrid-I.

Performance: As stated earlier, Hybrid-I uses the *overall query execution time* of workload Q, denoted as $ORunT(Q, Q_{pub})$, as an indicator of performance. Consequently, the objective function of WPP aims to minimize the overall execution time of a given query workload Q. The execution time of queries in Q over a hybrid cloud, given that queries in Q_{pub} are executed on a public cloud can be represented as follows:

$$Performance(Q, Q_{pub}) = ORunT(Q, Q_{pub}) = \max \begin{cases} \sum\limits_{q \in Q_{pub}} freq(q) \times runT_{pub}(q) \\ \sum\limits_{q \in Q - Q_{pub}} freq(q) \times runT_{priv}(q) \end{cases}$$

Note that $Q_{pub} \subseteq Q$, otherwise it is undefined. Additionally, $freq(q)$ denotes the access frequency of query $q \in Q$ and $runT_x(q)$ denotes the estimated running time of query $q \in Q$ at site x where x is either a public ($x = pub$) or private ($x = priv$) cloud. In practice, Hybrid-I uses the I/O size of the query execution plan selected for processing q at site x as a replacement for the execution time. The running time of a query q can be estimated based on the selected query plan T for site x (x is a public or private cloud) as follows:

$$runT_x(q) = runT_x(T) = \frac{\sum\limits_{\forall\ operator\ \rho \in T} inpSize(\rho) + outSize(\rho)}{w_x},$$

where $inpSize(\rho)$ and $outSize(\rho)$ denote the estimated input and output sizes of an operator $\rho \in T$. Additionally, weight w_x denotes the number of I/O operations that can be performed per unit time at site x. Note that $inpSize(\rho)$ and $outSize(\rho)$ can be computed using statistics accumulated over dataset R for an operator ρ.

Data Disclosure Risk: In Hybrid-I, the risk associated with storing the public side partition of data, namely R_{pub}, using the representations given in Rep, namely plaintext and bucketization, is estimated as follows:

$$Risk(R_{pub}, Rep) = \sum_{R_i \in R_{pub}, s \in Rep} sens(R_i, s),$$

where $sens(R_i, s)$ is the number of sensitive values contained in a data item $R_i \in R_{pub}$, which are stored under the representation, $s \in Rep$, on a public cloud. Finally, the formalization of WPP places a user defined upper bound, $DISC\_CONST$, on the amount of sensitive data that can be disclosed to a public cloud.

Resource Allocation Cost: Hybrid-I estimates the financial cost of utilizing public cloud services as follows:

$$Pricing(R_{pub}, Q_{pub}) = store(R_{pub}) + \sum_{q \in Q_{pub}} freq(q) \times proc(q),$$

where $store(R_{pub})$ represents the monetary cost of storing a subset $R_{pub} \subseteq R$ on a public cloud, $freq(q)$ denotes the access frequency of query $q \in Q$, and $proc(q)$ denotes the monetary cost associated with processing query q on a public cloud. Finally, the formalization of WPP incorporates a user defined parameter, PRA_CONST, which acts as an upper bound on the maximum allowable monetary cost that can be expended on storing and processing data on a public cloud.

5 Conclusions and Future Work

A hybrid cloud is well suited for users who want to balance the efficiency achieved through the distribution of computational workloads with the risk of exposing sensitive information, the monetary costs associated with acquiring public cloud services and the load generated on a private cloud as a result of processing some part of a workload. In this paper, we identified the criteria that have the greatest impact on the design of an effective hybrid cloud solution and we tabulated the applicability of these criteria to various hybrid cloud deployment models. Then, we formalized the workload partitioning problem as a mechanism for maximizing workload performance using the identified criteria. Finally, we described how existing systems could be derived from the general workload partitioning problem formalization by an instantiation of the appropriate design criteria.

As a part of our future work, we plan to expand on the design criteria we identified in this paper by including factors such as the processing capabilities of a public cloud, which also greatly affect the performance of a hybrid cloud application.

6 Acknowledgements

The work conducted at UT Dallas was partially supported by The Air Force Office of Scientific Research MURI-Grant FA-9550-08-1-0265 and Grant FA9550-12-1-0082, National Institutes of Health Grant 1R01LM009989, National Science Foundation (NSF) Grant Career-CNS-0845803, NSF Grants CNS- 0964350, CNS-1016343, CNS-1111529, and CNS-1228198, and Army Research Office Grant 58345-CS. The work conducted at UC Irvine was supported by the National Science Foundation under Grant No. 1118127.

References

[1] Hybrid Cloud. The NIST Definition of Cloud Computing. *National Institute of Science and Technology, Special Publication, 800-145*, 2011.

[2] K. Zhang, X–y. Zhou, Y. Chen, XF. Wang, and Y. Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *ACM Conference on Computer and Communications Security*, pages 515–526, 2011.

[3] K. Y. Oktay, V. Khadilkar, B. Hore, M. Kantarcioglu, S. Mehrotra, and B. Thuraisingham. Risk-Aware Workload Distribution in Hybrid Clouds. In *IEEE CLOUD*, pages 229–236, 2012.

[4] M. Lev-Ram. Why Zynga loves the hybrid cloud. `http://tech.fortune.cnn.com/2012/04/09/zynga-2/?iid=HP_LN`, 2012.

[5] L. Mearian. EMC's Tucci sees hybrid cloud becoming de facto standard. `http://www.computerworld.com/s/article/9216573/EMC_s_Tucci_sees_hybrid_cloud_becoming_de_facto_standard`, 2011.

[6] Accenture Technology Vision 2011 - The Technology Waves That Are Reshaping the Business Landscape. `http://www.accenture.com/us-en/technology/technology-labs/Pages/insight-accenture-technology-vision-2011.aspx`, 2011.

[7] M. R. Fouad, G. Lebanon, and E. Bertino. ARUBA: A Risk-Utility-Based Algorithm for Data Disclosure. In *Secure Data Management*, pages 32–49, 2008.

[8] S. Trabelsi, V. Salzgeber, M. Bezzi, and G. Montagnon. Data disclosure risk evaluation. In *CRiSIS*, pages 35–72, 2009.

[9] Using Data Encryption. `http://docs.amazonwebservices.com/AmazonS3/latest/dev/index.html?UsingEncryption.html`

[10] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In USENIX Security Symposium, pages 45–60. USENIX Association, 2008.

[11] H. Herodotou. Hadoop Performance Models. Technical Report CS-2011-05, Computer Science Department, Duke University.

[12] H. Hacigümüş, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.

Replicated Data Integrity Verification in Cloud

Raghul Mukundan
Department of Computer Science
Missouri University of Science and Technology
rmgq8@mst.edu

Sanjay Madria
Department of Computer Science
Missouri University of Science and Technology
madrias@mst.edu

Mark Linderman
Air Force Research Lab
Rome, NY
mark.linderman@rl.af.mil

Abstract

Cloud computing is an emerging model in which computing infrastructure resources are provided as a service over the Internet. Data owners can outsource their data by remotely storing them in the cloud and enjoy on-demand high quality applications and services from a shared pool of configurable computing resources. However, since data owners and cloud servers are not in the same trusted domain, the outsourced data may be at risk as the cloud server may no longer be fully trusted. Therefore, data integrity is of critical importance in such a scenario. Cloud should let either the owners or a trusted third party to audit their data storage without demanding a local copy of the data from owners. Replicating data on cloud servers across multiple data centers provides a higher level of scalability, availability, and durability. When the data owners ask the Cloud Service Provider (CSP) to replicate data at different servers, they are charged a higher fee by the CSP. Therefore, the data owners need to be strongly convinced that the CSP is storing all the data copies that are agreed upon in the service level contract, and the data-update requests issued by the customers have been correctly executed on all the remotely stored copies. To deal with such problems, previous multi copy verification schemes either focused on static files or incurred huge update costs in a dynamic file scenario. In this paper, we propose some ideas under a Dynamic Multi-Replica Provable Data Possession scheme (DMR-PDP) that prevents the CSP from cheating; for example, by maintaining fewer copies than paid for. DMR-PDP also supports efficient dynamic operations like block modification, insertion and deletion on data replicas over cloud servers.

1 Introduction

When users store data in the cloud, their main concern is whether the cloud can maintain the data integrity and data can be recovered when there is a data loss or server failure. Cloud service providers (CSP), in order to save storage cost, may tend to discard some data or data copies that is not accessed often, or mitigate data to second-level storage devices. CSPs may also conceal data loss due to management faults, hardware

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This publication has been cleared for public release, distribution unlimited by AFRL (case number 88ABW-2012-6360).

failures or attacks. Therefore, a critical issue in storing data at untrusted CSPs is periodically verifying whether the storage servers maintain data integrity; store data completely and correctly as stated in the service level agreement (SLA).

Data replication is a commonly used technique to increase the data availability in cloud computing. Cloud replicates the data and stores them strategically on multiple servers located at various geographic locations. Since the replicated copies look exactly similar, it is difficult to verify whether the cloud really stores multiple copies of the data. Cloud can easily cheat the owner by storing only one copy of the data. Thus, the owner would like to verify at regular intervals whether the cloud indeed possesses multiple copies of the data as claimed in the SLA. In general, cloud has the capability to generate multiple replicas when a data owner challenges the CSP to prove that it possesses multiple copies of the data. Also, it is a valid assumption that the owner of the data may not have a copy of the data stored locally. So, the major task of the owner is not only to verify that the data is intact but also to recover the data if any deletions/corruptions of data are identified. If the data owner during his verification using DMR-PDP scheme detects some data loss in any of the replicas in the cloud, he can recover the data from other replicas that are stored intact. Since, the replicas are to be stored at diverse geographic locations, it is safe to assume that a data loss will not occur at all the replicas at the same time.

Provable data possession (PDP) [2] is a technique to audit and validate the integrity of data stored on remote servers. In a typical PDP model, the data owner generates metadata/tag for a data file to be used later for integrity verification. To ensure security, the data owner encrypts the file and generates tags on the encrypted file. The data owner sends the encrypted file and the tags to the cloud, and deletes the local copy of the file. When the data owner wishes to verify the data integrity, he generates a challenge vector and sends it to the cloud. The cloud replies by computing a response on the data and sends it to the verifier/data owner to prove that multiple copies of the data file are stored in the cloud. Different variations of PDP schemes such as [2], [4], [6], [7], [9], [10], [11], [12], [15] were proposed under different cryptographic assumptions. But most of these schemes deal only with static data files and are valid only for verifying a single copy. A few other schemes such as [3], [5], [8], [13], [14] provide dynamic scalability of a single copy of a data file for various applications which mean that the remotely stored data can not only be accessed by the authorized users, but also be updated and scaled by the data owner.

In this paper, we propose a scheme that allows the data owner to securely ensure that the CSP stores multiple replicas. A simple way to make the replicas look unique and differentiable is using probabilistic encryption schemes. Probabilistic encryption creates different cipher texts each time the same message is encrypted using the same key. Thus, our scheme uses homomorphic probabilistic encryption to create distinct replicas/copies of the data file and BLS signatures [17] to create constant amount of metadata for any number of replicas. Probabilistic encryption encrypts all the replicas with the same key. Therefore, in our scheme the data owner will have to share just one decryption key with the authorized users and need not worry about CSP granting access to any of the replicas to the authorized users. The homomorphic property of the encryption scheme helps in efficient file updates. The data owner has to encrypt the difference between the updated file and the old file and send it to the cloud, which updates all the replicas by performing homomorphic addition on the file copies. Any authenticated data structure, e.g, Merklee Hash Trees and Skiplist can be used with our scheme to ensure that the cloud uses the right file blocks for data integrity verification. However, the ways to efficiently manage authenticated data structures in the cloud is not within the scope of our paper.

Organization: The rest of the paper is organized as follows. Overview of the related work is provided in Section 2 followed by the problem definition in Section 3, and a detailed description of our scheme in Section 4. Future work is discussed in Section 5.

2 Related Work

Ateniese et al. [2] were the first to define the Provable Data Possession (PDP) model for ensuring the possession of files on untrusted storages. They made use of RSA-based homomorphic tags for auditing outsourced data. However, dynamic data storage and multiple replica system are not considered in this scheme. In their subsequent work [12], they proposed a dynamic version which supports very basic block operations with limited functionality, and does not support block insertions. In [13], Wang et al. considered dynamic data storage in a distributed scenario, and proposed a challenge-response protocol which can determine data correctness as well as locate possible errors. Similar to [12], only partial support for dynamic data operation is considered. Erway et al. [14] extended the PDP model in [2] to support provable updates to stored data files using rank-based authenticated skip lists. However, efficiency of their scheme remains unclear and these schemes hold good only for verifying a single copy.

Curtmola et.al [1] proposed Multiple-Replica PDP (MR-PDP) scheme wherein the data owner can verify that several copies of a file are stored by a storage service provider. In their scheme, distinct replicas are created by first encrypting the data and then masking it with randomness generated from a Pseudo-Random Function (PRF). The randomized data is then stored across multiple servers. The scheme uses RSA signatures for creation of tags. But, their scheme did not address how the authorized users of the data can access the file copies from the cloud servers noting that the internal operations of the CSP are opaque and do not support dynamic data operations. Ayad F. Barsoum et al. [16] proposed creation of distinct copies by appending replica number to the file blocks and encrypting it using an encryption scheme that has strong diffusion property, e.g., AES. Their scheme supports dynamic data operations but during file updates, the copies in all the servers should be encrypted again and updated on the cloud. This scheme suits perfectly for static multiple replicas but proves costly in a dynamic scenario. BLS signatures are used for creating tags and authenticated data structures like Merklee Hash Trees are used to ensure right file blocks are used during verification. Authorized users of the data should know random numbers in [1] and replica number in [16] to generate the original file.

3 Dynamic Multi-Replica Provable Data Possession (DMR-PDP) Scheme

The cloud computing model considered in this work consists of three main components as illustrated in Figure 1: (i) a data owner that can be an individual or an organization originally possessing sensitive data to be stored in the cloud; (ii) space on its infrastructure to store the owne ave the right to access the remote data and s

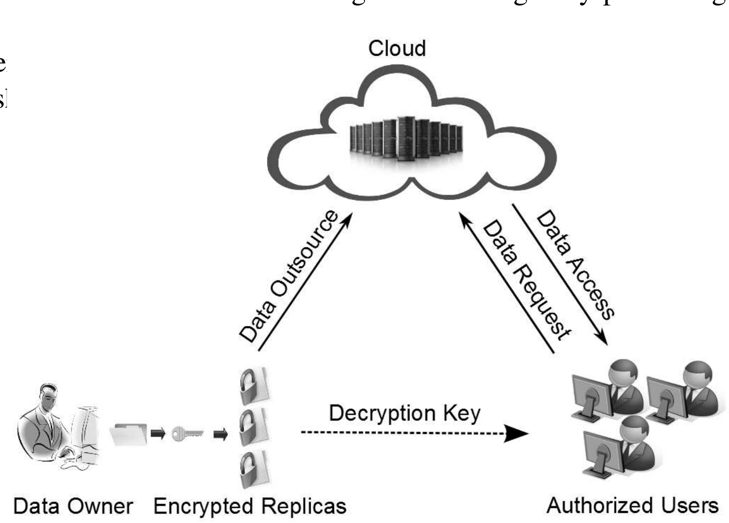


Figure 1: Cloud Computing Data Storage Model.

3.1 Problem Definition and Design Goals

More recently, many data owners relieve their burden of local data storage and maintenance by outsourcing their data to a CSP. CSP undertakes the data replication task in order to increase the data availability, durability and reliability but the customers have to pay for using the CSPs storage infrastructure. On the other hand, cloud customers should be convinced that the (1) CSP actually possesses all the data copies as agreed upon, (2) integrity of these data copies are maintained, and (3) the customers are receiving the service that they are paying for. Therefore, in this paper, we address the problem of securely and efficiently creating multiple replicas of the data file of the owner to be stored over untrusted CSP and then auditing all these copies to verify their completeness and correctness. Our design goals are summarized below:

1. Dynamic Multi-Replica Provable Data Possession (DMR-PDP) protocols should efficiently and securely provide the owner with strong evidence that the CSP is in possession of all the data copies as agreed upon and that these copies are intact.
2. Allowing the users authorized by the data owner to seamlessly access a file copy from the CSP.
3. Using only a single set of metadata/tags for all the file replicas for verification purposes.
4. Allowing dynamic data operation support to enable the data owner to perform block-level operations on the data files while maintaining the same level of data correctness assurance.
5. Enabling both probabilistic and deterministic verification guarantees.

3.2 Preliminaries and Notations

In this section, we provide details of the Bilinear mapping and Paillier Encryption schemes used in our present work.

1. Assume that F, a data file to be outsourced, is composed of a sequence of *m* blocks, i.e., F = $\{b_1, b_2,..,b_m\}$.
2. $F_i = \{b_{i1}, b_{i2},....,b_{im}\}$ represents the file copy *i*.
3. **Bilinear Map/Pairing:** Let G_1, G_2, and G_T be cyclic groups of prime order *a*. Let *u* and *v* be generators of G_1 and G_2, respectively. A bilinear pairing is a map e : G_1 x $G_2 \rightarrow G_T$ with the following properties:
   - Bilinear: $e(u_1u_2, v_1) = e(u_1, v_1) . e(u_2, v_1)$, $e(u_1, v_1v_2) = e(u_1, v_1) . e(u_1,v_2)$ $\forall$ $u_1, u_2 \in G_1$ and $v_1, v_2 \in G_2$
   - Non-degenerate: $e(u, v) \neq 1$
   - There exists an efficient algorithm for computing *e*
   - $e({u_1}^x, {v_1}^y) = e(u_1, v_1)^{xy}$ $\forall$ $u_1 \in G_1$; $v_1 \in G_2$, and x, y $\in Z_a$
4. H(.) is a map-to-point hash function: $\{0, 1\}^* \rightarrow G_1$.
5. **Homomorphic Encryption:** A homomorphic encryption scheme has the following properties.
   - $E(m_1 + m_2) = E(m_1) +_h E(m_2)$ where $+_h$ is a homomorphic addition operation.
   - $E(k*m) = E(m)^k$.

   where E(.) represents a homomorphic encryption scheme and m, m_1, m_2 are messages that are encrypted and k is some random number.

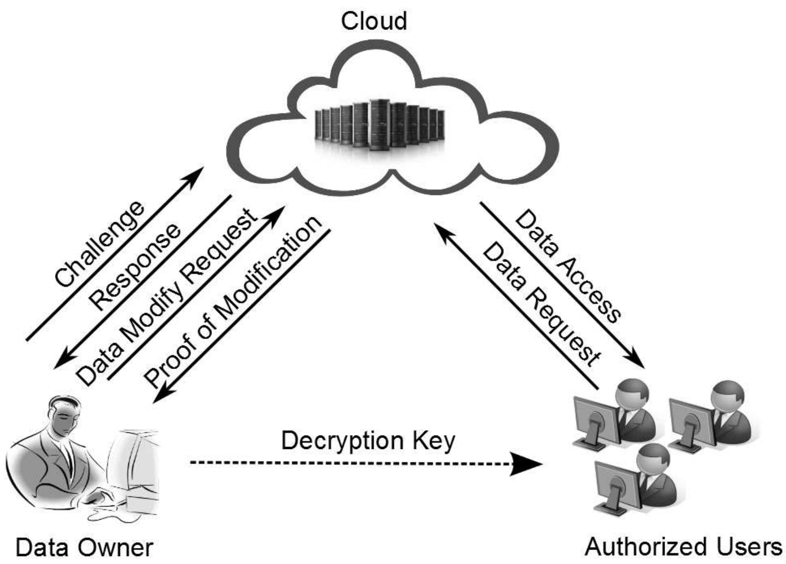


Figure 2: DMR-PDP Scheme

6. **Paillier Encryption:** Paillier cryptosystem is a homomorphic probabilistic encryption scheme. The steps are as follows.
   - Compute N = p * q and λ = LCM (p-1, q-1), where p, q are two prime numbers.
   - Select a random number g such that its order is a multiple of N and $g \in \mathrm{Z}_N{}^{2*}$.
   - Public key is (N, g) and secret key is λ, where N = p*q.
   - Cipher text for a message m is computed as c = $\mathrm{g}^m\ \mathrm{r}^N$ mod N^2 where r is a random number and $\mathrm{r} \in \mathrm{Z}_N{}^*$, $\mathrm{c} \in \mathrm{Z}_N{}^{2*}$ and $m \in \mathrm{Z}_N$.
   - Plain text is obtained by m = L(c^λ mod N^2) * (L(g^λ mod N^2))$^{-1}$ mod N.
7. Properties of public key g in Paillier Scheme
   - $g \in \mathrm{Z}_N{}^{2*}$.
   - If g = (1 + N) mod N^2, it has few interesting properties
     (a) Order of the value (1 + N) is N.
     (b) $(1 + \mathrm{N})^m \equiv (1 + \mathrm{mN})$ mod N^2. (1 + mN) can be used directly instead of calculating $(1 + \mathrm{N})^m$. This avoids the costly exponential operation during data encryption.

3.3 DMR-PDP Construction

In our approach, the data owner creates multiple encrypted replicas and uploads them on to the cloud. The CSP stores them on one or multiple servers located at various geographic locations. The data owner shares the decryption key with a set of authorized users. In order to access the data, an authorized user sends a data request to the CSP and receives a data copy in an encrypted form that can be decrypted using a secret key shared with the owner. The proposed scheme consists of seven algorithms: KeyGen, ReplicaGen, TagGen, Prove, Verify, PrepareUpdate and ExecUpdate. The overview of the communication involved in our scheme is shown in 2.

1. (pk, sk) $\leftarrow$ KeyGen(). This algorithm is run by the data owner to generate a public key *pk* and a private key *sk*. The data owner generates three sets of keys.
   (a) Keys for data tags : This key is used for generating tags for the data. The data owner selects a bilinear map *e* and selects a private key $\mathrm{l} \in \mathrm{Z}_a$. Public key is calculated as $\mathrm{y} = \mathrm{v}^l \in \mathrm{G}_2$.

(b) Keys for data : This key is used for encrypting the data and thereby creating multiple data copies. The data owner selects paillier public keys (N, g) with g = (1 + N) mod N^2 and secret key λ.

(c) PRF key : The data owner generates a PRF key Key_{PRF} which generates s numbers. These s numbers are used in creating s copies of the data. Each number is used in creating one data copy. Let $\{k_1, k_2,..,k_s\} \in Z_N{}^*$ be the numbers generated by the PRF key. Key_{PRF} is maintained confidentially by the data owner and hence the s numbers used in creating multiple copies are not known to the cloud.

2. $\{F_i\}_{1 \leq i \leq s} \leftarrow$ ReplicaGen (s, F). This algorithm is run by the data owner. It takes the number of replicas s and the file F as input and generates s unique differentiable copies $\{F_i\}_{1 \leq i \leq s}$. This algorithm is run only once. Unique copies of each file block of file F is created by encrypting it using a probabilistic encryption scheme, e.g., Paillier encryption scheme.

   Through probabilistic encryption, encrypting a file block s times yields s distinct cipher texts. For a file F = $\{b_1, b_2,..,b_m\}$ multiple data copies are generated using Paillier encryption scheme as $F_i = \{(1+N)^{b_1}(k_i r_{i1})^N, (1+N)^{b_2}(k_i r_{i2})^N,.., (1+N)^{b_m}(k_i r_{im})^N\}_{1 \leq i \leq m}$. Using Paillier's properties the above result can be rewritten as $F_i = \{(1+b_1 N)(k_i r_{i1})^N, (1+b_2 N)(k_i r_{i2})^N,.., (1+b_m N)(k_i r_{im})^N\}_{1 \leq i \leq m}$, where i represents the file copy number, k_i represents the numbers generated from PRF key Key_{PRF} and r_{ij} represents any random number used in Paillier encryption scheme. k_i is multiplied by a random number r_{ij} and the product is used for encryption. The presence of k_i in a file block identifies which file copy the file block belongs to. All these file copies yield the original file when decrypted . This allows the users authorized by the data owner to seamlessly access the file copy received from the CSP.

3. $\phi \leftarrow$ TagGen (sk, F). This algorithm is run by the data owner. It takes the private key sk and the file F as input and outputs the tags ϕ. We use BLS signature scheme to create tags on the data. BLS signatures are short and homomorphic in nature and allow concurrent data verification, which means multiple data blocks can be verified at the same time. In our scheme, tags are generated on each file block b_i as $\phi_i = (H(F) . u^{b_i N})^l \in G_1$ where $u \in G_1$ and $H(.) \in G_1$ represents hash value which uniquely represents the file F. The data owner sends the tag set $\phi = \{\phi_i\}_{1 \leq i \leq m}$ to the cloud.

4. $P \leftarrow$ Prove (F, ϕ, *challenge*). This algorithm is run by the CSP. It takes the file replicas of file F, the tags ϕ and *challenge* vector sent by the data owner as input and returns a proof P which guarantees that the CSP is actually storing s copies of the file F and all these copies are intact. The data owner uses the proof P to verify the data integrity. There are two phases in this algorithm:

   (a) **Challenge:** In this phase the data owner challenges the cloud to verify the integrity of all outsourced copies. There are two types of verification schemes:

      i. Deterministic - here all the file blocks from all the copies are used for verification.
      ii. Probabilistic - only a few blocks from all the copies are used for verification. A Pseudo Random Function key (PRF) is used to generate random indices ranging between 1 and m. The file blocks from these indices are used for verification. In each verification a percentage of the file is verified and it accounts for the verification of the entire file.

   At each challenge, the data owner chooses the type of verification scheme he wishes to use. If the owner chooses the deterministic verification scheme, he generates one PRF key, Key_1. If he chooses the probabilistic scheme he generates two PRF keys, Key_1 and Key_2. PRF keyed with Key_1 generates c ($1 \leq c \leq m$) random file indices which indicates the file blocks that CSP should use for verification. PRF keyed with Key_2 generates s random values and the CSP should use each of these random numbers for each file copy while computing the response. The data owner sends the generated keys to the CSP.

(b) **Response:** This phase is executed by the CSP when a challenge for data integrity verification is received from the data owner. Here, we show the proof for probabilistic verification scheme (the deterministic verification scheme also follows the same procedure). The CSP receives two PRF keys, Key_1 and Key_2 from the data owner. Using Key_1, CSP generates a set {C} with c ($1 \leq c \leq m$) random file indices ($\{C\} \in \{1, 2,..,m\}$), which indicate the file blocks that CSP should use for verification. Using Key_2, CSP generates 's' random values T = $\{t_1, t_2,..,t_s\}$. The cloud performs two operations. One on the tags and the other on the file blocks.

i. Operation on the tags: Cloud multiplies the file tags corresponding to the file indices generated by PRF key Key_1.

$$\begin{aligned}\sigma &= \prod_{j\in C}(H(F).u^{b_jN})^l \\ &= \prod_{j\in C} H(F)^l.\prod_{j\in C} u^{b_jNl} \\ &= H(F)^{cl}.u^{Nl\sum\limits_{j\in C}(b_j)}\end{aligned}$$

ii. Operation on the file blocks: The cloud first takes each file copy and multiplies all the file blocks corresponding to the file indices generated by PRF key Key_1. The product of each copy is raised to the power the random number generated for that copy by the PRF key Key_2. The result of the above operation for each file copy i is given by $(\prod\limits_{j\in C}(1+N)^{b_j}(k_i r_{ij})^N)^{t_i}$ mod N^2. The CSP then multiplies the result of each copy to get the result

$$\begin{aligned}\mu &= \prod_{i=1}^{s}(\prod_{j\in C}(1+N)^{b_j}(k_i r_{ij})^N)^{t_i} \\ &= \prod_{i=1}^{s}(\prod_{j\in C}(1+N)^{b_j t_i}\prod_{j\in C}(k_i r_{ij})^{Nt_i}) \\ &= \prod_{i=1}^{s}((1+N)^{t_i\sum\limits_{j\in C} b_j}\prod_{j\in C}(k_i r_{ij})^{Nt_i}) \\ &= (\prod_{i=1}^{s}(1+N)^{t_i\sum\limits_{j\in C} b_j})(\prod_{i=1}^{s}((k_i)^{ct_iN}\prod_{j\in C}(r_{ij})^{Nt_i})) \\ &= ((1+N)^{\sum\limits_{i=1}^{s} t_i\sum\limits_{j\in C} b_j})(\prod_{i=1}^{s}(k_i)^{ct_iN})(\prod_{i=1}^{s}\prod_{j\in C}(r_{ij})^{Nt_i})\end{aligned}$$

Using properties of Paillier scheme, the above equation can be rewritten as

$$\mu = (1+N\sum_{i=1}^{s}(t_i)\sum_{j\in C}(b_j))(\prod_{i=1}^{s}(k_i)^{Nct_i})(\prod_{i=1}^{s}(\prod_{j\in C}(r_{ij})^{t_iN}))$$

The CSP sends σ and μ mod N^2 values to the data owner.

5. $\{1, 0\} \leftarrow$ Verify (pk, P). This algorithm is run by the data owner. It takes as input the public key pk and the proof P returned from the CSP, and outputs 1 if the integrity of all file copies is correctly verified or 0 otherwise. After receiving σ and μ values from the CSP, the data owner does the following:

Owner		**CSP**
1. Calculates $\Delta b_j = b_j' - b_j$. 2. Encrypts Δb_j using Paillier encryption. $E(\Delta b_j) = (1 + \Delta b_j N)\, r^N$, where *r* is some random number. 3. Calculates the new file tag for b_j', $\phi' = (H(F)\, u^{b'_j N})^l$. 4. Generates PRF keys Key_1, Key_2 to verify the correctness of modify operation.		
	$\xrightarrow{\langle Id_F,\ \text{modify},\ j,\ E(\Delta b_j),\ \phi' \rangle,\ Key_1,\ Key_2}$	
		5. Performs homomorphic addition operation $E(b_j') = E(\Delta b_j) * E(b_j)$ on all the file copies. 6. Deletes the old tag and replaces it with the new tag ϕ'. 7. Calculates a response μ, σ.
	$\xleftarrow{\mu,\ \sigma}$	
8. Calculates *v* and *d* 9. Verifies if $\mu \bmod v \equiv 0$ and checks if $(H(F)^c\, u^{dN})^l = \sigma$.		

Figure 3: Block modification operation in the DMR-PDP scheme

(a) calculates $v = (\prod_{i=1}^{s} (k_i)^{t_i c N})$ and $d = Decrypt(\mu)/(\sum_{i=1}^{s} k_i)$. This can be calculated from the values generated from Key_{PRF} and *c*.

(b) checks if $\mu \bmod v \equiv 0$. This ensures that the cloud has used all the file copies while computing the response.

(c) checks if $(H(F)^c\, u^{dN})^l = \sigma$. This ensures that the CSP has used all the file blocks while computing the response. If options b and c are satisfied, it indicates that the data stored by the owner in the cloud is intact and the cloud has stored multiple copies of the data as agreed in the service level agreement.

6. *Update* $\leftarrow$ PrepareUpdate (). This algorithm is run by the data owner to perform any operation on the outsourced file copies stored by the remote CSP. The output of this algorithm is an *Update* request. The data owner sends the *Update* request to the cloud and will be of the form $\langle Id_F$, BlockOp, j, b_i', $\phi' \rangle$, where Id_F is the file identifier, BlockOp corresponds to block operation, j denotes the index of the file block, b_i' represents the updated file blocks and ϕ' is the updated tag. BlockOp can be data modification, insertion or delete operation.

7. (F', ϕ') $\leftarrow$ ExecUpdate (F, ϕ, *Update*). This algorithm is run by the CSP where the input parameters are the file copies F, the tags ϕ, and *Update* request (sent from the owner). It outputs an updated version of all the file copies F' along with updated signatures ϕ'. After any block operation, the data owner runs the challenge protocol to ensure that the cloud has executed the operations correctly. The operation in *Update* request can be modifying a file block, inserting a new file block or deleting a file block.

   (a) **Modification:** Data modification is one of the most frequently used dynamic operations. The data modification operation in DMR-PDP scheme is shown in Figure 3.

   (b) **Insertion:** In the block insertion operation, the owner inserts a new block after position *j* in a file. If the file *F* had *m* blocks initially, the file will have *m+1* blocks after the insert operation.

The file block insertion operation is shown in Figure 4.

(c) **Deletion:** Block deletion operation is the opposite of the insertion operation. When one block is deleted, indices of all subsequent blocks are moved one step forward. To delete a specific data block at position *j* from all copies, the owner sends a delete request <Id_F, delete, j, null, null> to the cloud. Upon receiving the request, the cloud deletes the tag and the file block at index *j* in all the file copies.

Owner | **CSP**

1. Encrypts the new file block *s* times
2. Creates tag ϕ for the new file block
3. Generates PRF keys Key_1, Key_2 to verify the correctness of insert operation.

$\xrightarrow{<\mathrm{Id}_F,\ \text{insert, j, } s \text{ file blocks, } \phi>\ ,\ Key_1,\ Key_2}$

4. Inserts the new file block at location *j*
5. Stores the new tag ϕ.
6. Calculates a response μ, σ.

$\xleftarrow{\mu,\ \sigma}$

7. Calculates *v* and *d*
8. Verifies if $\mu \bmod v \equiv 0$ and checks if $(\mathrm{H(F)}^c\ \mathrm{u}^{dN})^l = \sigma$.

Figure 4: Block insertion operation in the DMR-PDP scheme

4 Conclusions and Future Work

In this paper, we have discussed work related to the replicated data integrity preservation in a cloud environment and presented a Dynamic Multi-Replica Provable Data Possession scheme (DMR-PDP) to periodically verify the correctness and completeness of multiple data copies stored in the cloud. Our scheme also supports dynamic data update operations. All the data copies can be decrypted using a single decryption key, thus providing a seamless access to the data's authorized users. This scheme can be extended for multiple versions where only deltas can be stored in the cloud and owner can save on storage cost. Currently, we are implementing the proposed scheme for evaluating it in a real cloud platform using different performance metrics and comparing it with some of the existing methods. We also plan to extend this scheme for secure multi-version data where only one original and multiple deltas can be stored in the cloud.

References

[1] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," in 28th IEEE ICDCS, 2008, pp. 411-420.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security, New York, NY, USA, 2007, pp. 598-609.

[3] G. Ateniese, R. D. Pietro, L. V. Mancin, and G. Tsudik, "Scalable and efficient provable data possession," in SecureComm âĂŹ08: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks, New York, NY, USA,2008, pp. 1-10.

[4] Y. Deswarte, J.-J. Quisquater, and A. SaÄśdane, "Remote integrity checking," in 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS), S. J. L. Strous, Ed., 2003, pp. 1-11.

[5] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” in CCS âĂŹ09: Proceedings of the 16th ACM Conference on Computer and Communications Security, New York, NY, USA, 2009, pp. 213-222.

[6] D. L. G. Filho and P. S. L. M. Barreto, “Demonstrating data possession and uncheatable data transfer,” Cryptology ePrint Archive, Report 2006/150, 2006.

[7] P. Golle, S. Jarecki, and I. Mironov, “Cryptographic primitives enforcing communication and storage complexity,” in FC’02: Proceedings of the 6th International Conference on Financial Cryptography, Berlin, Heidelberg, 2003, pp. 120-135.

[8] Z. Hao, S. Zhong, and N. Yu, “A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability,” IEEE Transactions on Knowledge and Data Engineering, vol. 99, no. PrePrints, 2011.

[9] E. Mykletun, M. Narasimha, and G. Tsudik, “Authentication and integrity in outsourced databases,” Trans. Storage, vol. 2, no. 2, 2006.

[10] F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, “Efficient remote data possession checking in critical information infrastructures,” IEEE Trans. on Knowl. and Data Eng., vol. 20, no. 8, 2008.

[11] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, “Auditing to keep online storage services honest,” in HOTOS’07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems, Berkeley, CA, USA, 2007, pp. 1-6.

[12] M. A. Shah, R. Swaminathan, and M. Baker, “Privacy-preserving audit and extraction of digital contents,” Cryptology ePrint Archive, Report 2008/186, 2008.

[13] C. Wang, Q. Wang, K. Ren, and W. Lou, “Ensuring data storage security in cloud computing,” Cryptology ePrint Archive, Report 2009/081, 2009, http://eprint.iacr.org/.

[14] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, “Enabling public verifiability and data dynamics for storage security in cloud computing,” in ESORICSâĂŹ09: Proceedings of the 14th European Conference on Research in Computer Security, Berlin, Heidelberg, 2009, pp. 355-370.

[15] K. Zeng, “Publicly verifiable remote data integrity,” in Proceedings of the 10th International Conference on Information and Communications Security, ser. ICICS ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 419-434.

[16] A. F. Barsoum and M. A. Hasan, “On verifying dynamic multiple data copies over cloud servers,” Cryptology ePrint Archive, Report 2011/447, 2011, 2011, http://eprint.iacr.org/

[17] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” in ASIACRYPT ’01:Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security, London, UK, 2001, pp. 514-532.

Engineering Security and Performance with Cipherbase

Arvind Arasu Spyros Blanas Ken Eguro Manas Joglekar Raghav Kaushik
Donald Kossmann Ravi Ramamurthy Prasang Upadhyaya Ramarathnam Venkatesan

Abstract

Cipherbase is a full-fledged relational database system that leverages novel customized hardware to store and process encrypted data. This paper outlines the space of physical design options for Cipherbase and shows how application developers can implement their data confidentiality requirements by specifying the encryption method for static data storage and the acceptable information leakage for runtime data processing. The goal is to achieve a physical database design with the best possible performance that fulfills the application's confidentiality requirements.

1 Introduction

Data confidentiality is one of the main concerns of users of modern information systems. Techniques to protect data against curious attackers are particularly important for users of public cloud services [7]. For instance, a biologist who has carried out in-vitro experiments over several years and stores the results for further analysis in a public database cloud service (e.g., [9]) wants to make sure that her competitor does not have access to her results before she was able to publish them. Data confidentiality, however, is also critical in private clouds in which competitors may pay database administrators of a company to steal confidential business secrets.

Cipherbase is an extension of Microsoft SQL Server, specifically designed to help organizations leverage an efficient "database-as-a-service" while protecting their data against "honest-but-curious" adversaries. In particular, Cipherbase can protect the data against administrators that have root access privileges on the database server processes and the machines that run these processes. Administrators need these access privileges to do their job such as creating indexes, repartitioning the data, applying security patches to the machines, etc. However, these administrators do not need to see and interpret the data stored in the databases of those machines. Cipherbase features a novel hardware / software co-design that leverages customized hardware (based on FPGAs [5]) to securely decrypt, process, and re-encrypt data without giving externals who have access to the system a way to sniff the corresponding plaintext. One particular feature of Cipherbase is that it supports *configurable security*. Not all data stored in a database is sensitive. For instance, a great deal of Master data such as names of countries or exchange rates are public knowledge and need not be protected against curious attackers. Some data such as vendor addresses might be confidential, but a weak encryption might be sufficient. Information leakage of that data is embarrassing, but not a disaster. Other information such as customer PII (personally identifiable information) needs to be strongly encrypted. Cipherbase allows the specification of the confidentiality requirements of all data in a declarative way while preserving the general purpose functionality of a database system.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

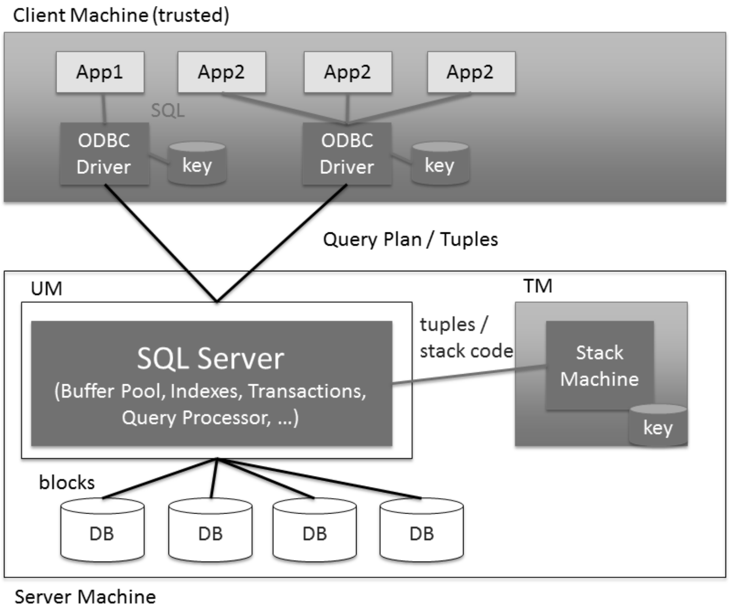


Figure 1: Cipherbase Architecture

1.1 System Overview

In this section, we give a brief overview of the Cipherbase system and contrast it with related work [3, 10](for more details on the system refer to [2]). The architecture of Cipherbase is shown in Figure 1. In a nutshell, Cipherbase extends all major components of Microsoft SQL Server. It extends the ODBC driver to keep encryption keys; this way, all encryption and decryption is transparent and applications only see clear text values so that they need not be rewritten. The SQL Server transaction and query processors at the server side are extended to make use of the trusted machine (referred to as TM in Figure 1) to process strongly encrypted data. The bulk of the SQL Server code is unchanged and runs on commodity servers (e.g. x86 processors), referred to as UM for untrusted machine in Figure 1. The TM is equivalent to an "oracle" that can compute on encrypted data without revealing the cleartext. The TM is implemented as a stack machine (Figure 1) which supports the ability to evaluate arbitrary predicates on encrypted data. The encryption keys are also stored in the TM in a tamper-proof manner and thus plaintext corresponding to encrypted data is not visible in the UM. This way, Cipherbase is able to support any kind of SQL query or update and preserve transactional semantics using the well-tested SQL Server code base. This allows us to maintain confidentiality by processing operations on encrypted data in the trusted machine in a fine-grained way whenever necessary.

In order to enable an efficient database-as-a-service, we need the ability to operate on encrypted data in the cloud. While generic homomorphic encryption techniques that enable arbitrary computation on encrypted data [6] are not yet practical, there are encryption techniques that enable executing different database operations directly on the encrypted data. For instance, if two columns are encrypted using deterministic encryption, the join can be evaluated by evaluating the join on the corresponding encrypted columns. CryptDB [10] is a recent system that exploits such partial homomorphic properties of various encryption schemes to support a subset of SQL queries. However, this approach has important limitations. First, CryptDB cannot handle generic and ad-hoc SQL queries. For instance, there is no known partial homomorphic encryption scheme that can handle even simple aggregates such as $price * (1.0 - discount)$. Thus, any ad-hoc query that computes such aggregates over a table needs to ship the entire table to a trusted proxy in order to decrypt the data to evaluate the aggregate (which would diminish the cost-benefit trade-offs in using the cloud in the first place). Second, even for the class of operations supported in CryptDB, the security is not configurable. For instance, if a query requires to sort a particular column, CryptDB would reorganize the column to be stored using order preserving encryption [4]. In contrast, Cipherbase provides *orthogonality* - it allows an application developer to choose an encryption policy independent of the query workload - a query can perform operations on strongly encrypted data (such as sorting a column) by leveraging the TM.

Our fine-grained integration of the UM/TM is also fundamentally different from the loosely coupled design of TrustedDB [3], that combines an IBM secure co-processor (SCP) and a commodity server (TrustedDB runs a lightweight SQLite database on SCP and a more feature-rich MySQL database on the commodity server). The fine-grained integration in Cipherbase enables a smaller footprint for the trusted hardware module as well as novel optimizations. For instance, if all columns were encrypted, the loosely coupled approach is constrained to use the limited computational resources at the TM for functionality that does not depend on encryption (e.g., disk spills in hash join). In contrast, Cipherbase adopts a tightly coupled design in which only the core primitives from each module of the database system that need to operate on encrypted data are factored and implemented in the TM. Interestingly, a small class of primitives involving encryption, decryption, and expression evaluation (see [2] for more details) suffices to support query processing, concurrency control, and other database functionality. For the hash join example, the hash join operator uses the TM for computing hashes and checking equality. The rest of hash join logic including memory management, writing hash buckets to disk, and reloading them all run in the UM (for a more detailed comparison with Trusted DB see [2]).

In terms of security, the TM is equivalent to an "oracle" that can compute on encrypted data without revealing the clear-text. Therefore, the information revealed is the results of operations used during query processing - a filter operator reveals identities of records satisfying the filter predicate, a sort operator reveals the ordering of records and a join operator reveals the join graph. The security of our basic system is similar to the security offered by CryptDB [10]. However, there are key differences. Even when computation is performed on a column, we only reveal information about the subset over which computation happens. For instance, if a subset of a table is sorted, we only reveal the ordering for the subset; in contrast, CryptDB reveals the ordering of the whole column. Second, we do not change the data storage, whereas CryptDB changes the data on disk to use weaker encryption and hence reveals information even to a weaker adversary who only has access to the disk.

As mentioned earlier, Cipherbase supports configurable security - it allows the specification of the confidentiality requirements of all data in a declarative way. More importantly, Cipherbase exploits this information to optimize query and transaction processing. For instance, public data can be processed in the untrusted machine (using conventional hardware) in the same way and as efficiently as with the existing SQL Server engine. Queries and transactions that involve public and confidential data are executed in both the UM and TM, thereby exploiting the UM as much as possible, minimizing use of the expensive TM, and minimizing data movement between the UM and TM. However, the act of query processing can itself leak information for an adversary who can monitor the "access-patterns" of query evaluation. In Cipherbase we take first steps in addressing this problem and provide mechanisms that can be used to mitigate this dynamic information leakage.

The purpose of this paper is to give details of the language extensions that allow users to specify the confidentiality requirements of their data (for more details on the system architecture and implementation refer to [2]). It turns out that confidentiality needs to be defined along two dimensions: (1) *static security* for encrypting the data stored in the disk (Section 3) and (2) *runtime security* to constrain the choice of algorithms to process data in motion because different algorithms imply different kinds of information leakage (Section 4). Furthermore, this paper outlines a number of tuning techniques (Section 5) that help to improve the performance of processing confidential data in a system like Cipherbase. The goal is to achieve the confidentiality that is needed at the lowest possible cost.

2 Static Data Security

Cipherbase supports a variety of different encryption techniques in order to meet a broad spectrum of privacy needs and allow users to tune their physical design for better performance. If privacy were the only concern, then all data should be encrypted in a strong and probabilistic way (e.g., using AES in CBC mode). While Cipherbase is able to support such a setting and execute any SQL query and update statement on such strongly encrypted data, the performance would be poor in many situations and the overhead would be much worse than necessary to fulfill the confidentiality needs (as mentioned earlier, for instance, some data need not be encrypted because it is public). We are currently integrating the following encryption techniques into Cipherbase:

- AES in ECB mode: This is a deterministic encryption technique which allows processing equality predicates on the encrypted data without decrypting the data. That is, any equality predicate can be processed in the UM entirely without shipping and processing any data in the TM.
- ROP [4]: This is an order-preserving encryption technique which allows processing range predicates, sorting, and Top N operations entirely in the UM. Aggregation or any kind of arithmetic or SQL intrinsic functions, however, must be carried out in the TM.
- Homomorphic: For certain arithmetic operators (e.g., addition and multiplication), practical encryption techniques are known. Generic homomorphic encryption [6] that, for instance, supports both addition and multiplication, however, is still not practical.
- AES in CBC mode: This is a strong, probabilistic encryption technique. Any kind of operation on this data needs to be carried out in the TM.

Our approach is based on the observation that data confidentiality needs are a *static* property of an application and do not depend on the query and update workload. For instance, compliance regulations might be the reason to encrypt some data strongly (e.g., the disease of patients in a health care domain) and some data not at all (e.g., cities of patients). These regulations are known statically and do not change as a result of executing the application. As a result, the data confidentiality needs should be specified in the schema as part of the SQL DDL by annotating the columns of a table with the encryption scheme that should be used to protect that data. The SQL query language and DML need not be changed. This design to specify confidentiality statically is in contrast to the design of CryptDB [10], a related database system for managing confidential data which adapts the encryption technique dynamically to the needs of the query workload. As discussed in Section 2, if processing of a query requires an order-preserving encryption technique and the column is currently strongly encrypted, then CryptDB will degrade the encryption scheme as a side-effect of processing this query. This design may result in performance jitter (reorganizing a whole table while processing, e.g., a fairly simple range query) and in unintended information leakage (degrading the encryption).

Figure 2 gives an example of a simple schema for patient and disease information. All patient information is considered to be confidential. The patient names are strictly confidential so that the schema of Figure 2 specifies that they be encrypted using AES. As *names* are unique in the *Patient* table, deterministic encryption (ECB mode) is sufficient. *Patient.age* is less critical; in particular, if the age cannot be associated to a name. As a result, *Patient.age* can be encrypted using a weaker, order-preserving technique such as ROP [4]. Disease information is public and the diagnosis information is again highly confidential because it belongs to patients.

As shown in Figure 2, any kind of integrity constraint can be implemented, independent of the encryption technique. Just as for regular query processing, however, the choice of the encryption technique impacts the performance of maintaining integrity constraints. In the schema of Figure 2, for instance, the `check` constraint of *Patient.age* can be validated by the UM without decrypting the patient information whenever a new patient is inserted or updated because an order-preserving encryption technique was chosen. If the age were encrypted using AES, then the TM would have to check the predicate of the integrity constraint.

A number of scenarios are conceivable to encrypt primary and foreign keys and test for referential integrity. Figure 2 shows two scenarios. First, *Diagnosis.patient* is encrypted in a stronger way than *Patient.name*; i.e., probabilistic encryption in CBC mode vs. deterministic encryption in ECB mode. The

```
create table Patient (
 name      :  VARCHAR(50)  AES_ECB primary key,
 age       :  VARCHAR(50)  ROP check >= 0);

create table Disease (
 name      :  VARCHAR(50)  primary key,
 descr     :  VARCHAR(256));

create table Diagnosis (
 patient   :  VARCHAR(50)  AES_CBC references Patient,
 disease   :  VARCHAR(50)  AES_ECB references Disease,
 date      :  DATE check not null);
```

Figure 2: Specifying Confidentiality Needs

motivation for using a probabilistic encryption technique for *Diagnosis.patient* might be that some patients are sick more often and we would like to hide this fact from a potential attacker. Doing so comes at a cost: The predicate evaluation of every *Patient* $\bowtie$ *Disease* needs to be carried out by the TM. Only if key and foreign key are encrypted using the same technique and this technique is deterministic (e.g., AES in ECB mode), a join can be carried out entirely in the UM without decrypting the join keys. As a result, application developers should carefully choose the encryption techniques of foreign keys and only diverge from the encryption technique of the primary key if absolutely necessary.

The second scenario shown in Figure 2 involves the *Diagnosis.disease* / *Disease.name* foreign key / key relationship. In this case, the foreign key is encrypted and the referenced primary key is not encrypted. Again, the motivation for encrypting the foreign key here might be that no information of the occurrence of certain diseases should be leaked to a potential attacker. Again, this situation of having different encryption techniques for join keys results in expensive *Diagnosis* $\bowtie$ *Patient* joins because the join predicate needs to be evaluated in the TM, thereby decrypting *Diagnosis.disease*. Curiously, in this example, better performance can be achieved by encrypting *Disease.name* using AES in ECB mode. That is, it might be beneficial to encrypt data for performance reasons even though the data is public and does not need to be protected. In general, there are many different ways to encrypt foreign keys and primary keys with different performance implications. [8], for instance, discusses an approach to probabilistically encrypt primary keys and foreign keys in concert in order to effect efficient joins without decrypting the keys.

3 Runtime Data Security

The previous section discussed encryption techniques and their performance implications to protect the data stored in the disk. This section discusses the performance implications in *runtime data security*. To motivate this discussion, the following simple example illustrates how the processing of data can leak information even if the UM never sees any unencrypted data.

Patient	*Disease*
Alice	!@#$xyz
Bob	@%^abc
Chen	*&#pqr
Dana	(p#z~94

Figure 3: Sample Diagnosis Table

Example 1: Figure 3 shows an example *Diagnosis* table. For brevity, the *date* column is not shown. To illustrate this example, it is assumed that the *patient* column is not encrypted and the *disease* column is strongly encrypted. That is, patient names are public information, but it should not be revealed which patients have been diagnosed with which disease. Now, consider a query that asks for the number of patients who have been diagnosed with a particular disease (e.g., AIDS). The predicate of this query needs to be executed by the TM because the *disease* column is encrypted. Nevertheless, a naïve execution of this query in which each tuple is checked individually by the TM can reveal information to a system administrator who is able to monitor the query execution. The naive execution of the filter operator would pass each encrypted tuple to the TM, which would decrypt the tuple and evaluate the predicate and return either true or the tuple again (which contains the patient name in cleartext). Thus, a system administrator who can monitor the communication between the TM and UM can: 1) infer that the multiple patients output by the filter operator have the same disease 2) infer the disease of the patients output by the filter operator (if he has additional background information that the disease in question is AIDS). Note that this is despite the fact that the column is stored strongly encrypted and never decrypted outside the TM.

The key observation is that query processing techniques and algorithms leak certain kinds of information (particularly when the schema involves a mix of both cleartext attributes and encrypted attributes). Cipherbase allows application developers to specify which kind of information may be leaked as a part of query processing by annotating the schema in the same way as specifying static data security. For instance, the designer of the schema of the *Diagnosis* table could specify that no dynamic information leakage is allowed for the *disease* column. As a result, Cipherbase would apply an appropriate algorithm to evaluate a query that asks for the number of AIDS patients.

Figure 4 lists the options that Cipherbase supports for runtime data security (similar to static security these knobs are specified on a per-column basis). The higher the confidentiality requirements, the more constrained the choice of query processing techniques and the more work needs to be carried out in the TM. In other words, the higher the confidentiality requirements, the worse the performance in many situations. We briefly describe the different levels and their performance implications in the remainder of this section (see [2] for a more detailed discussion on the implementation of the levels).

Dynamic Leakage	*Computation in TM*
Default	Expressions
Card	Expressions, defer & encrypt output
Blob	Whole operators, process blocks

Figure 4: Confidentiality Levels for Runtime Data Security

Default: This is the default level of runtime security that protects data on disk and in addition, it protects the data against attackers who can read the main memory contents of the database server machine. As a result, this level requires that data be kept in main memory in an encrypted form and may only be decrypted on the fly (in the TM) for query processing. This level requires the use of a trusted hardware (i.e., TM) to process, say, predications or aggregates on encrypted data, but it allows the use of the naïve algorithm to evaluate predicates of tuples. In other words, it does not protect against the monitoring attack described in Example 1.

Cardinality: This level only reveals the cardinality of intermediate and final query results (that include the sensitive column). So, if a query asks for all patients that have been diagnosed with AIDS, the attacker would be able to learn the total number of patients who have been diagnosed with AIDS, but the attacker would not be able to infer the name of a single AIDS patient. To implement this level, Cipherbase *defers* the evaluation of a predicate for a tuple [2]. In Example 1, for instance, if Alice was diagnosed with AIDS and the TM is asked to process Alice, then the TM would not immediately return Alice as a match when probed with Alice. Instead, the TM would remember all matching tuples and start returning matching tuples (in which all columns including any plain text columns are encrypted) at a later point. In addition, the order of the returned tuples is randomized. For instance, it could return Alice (in an encrypted form) when processing the tuple in the TM corresponding to Dana.

Blob: This level is essentially equivalent to storing the column as a blob. To implement this level, Cipherbase needs to implement whole operators of the relational algebra in the TM. To process queries, blocks of tuples that are encrypted as a whole are shipped to the TM. The TM decrypts these blocks and then carries out bulk operations (e.g., partitioning for a hash join) on these tuples. Furthermore, the TM returns only encrypted blocks of tuples (possibly even the empty block of tuples) so that no information can be inferred by observing the data returned from the TM. This option could have significant performance penalties for more complex queries (and could also involve more significant processing in the client). Even in the case of Example 1, this option has to return a constant number of blocks as output independent of the selectivity of the predicate (to ensure that the cardinality of the filter operator is not revealed) and thus, this option must be only used only for columns that require high confidentiality.

As with the case of static security, the options for runtime data security can also be specified declaratively as DDL annotations. For the example database shown in Figure 3, the annotation would be as follows.

```
create table Diagnosis (
name    :  VARCHAR(50) references Patient,
disease :  VARBINARY  AES_CBC  BLOB references Disease);
```

4 Other Tuning Options

Specifying the encryption method and the degree of information leakage as described in the previous two sections are the most critical decisions for the physical design of a database like Cipherbase. This section discusses additional considerations for a good physical design in Cipherbase.

Horizontal clustering: Some schemas involve a great deal of flags (e.g., *is vegetarian*) or small fields that can be represented using a few bits (e.g., *status*). Encrypting each of these fields individually is wasteful. For instance, if a flag (1 bit) is encrypted using AES (128 bits), then the size of the database can grow by two orders of magnitude. One way to reduce this space overhead without sacrificing confidentiality is to cluster a set of attributes of a row and encrypt them together into a single (128 bit) cipher.

Vertical clustering: An alternative to the horizontal clustering and encryption of several fields within a row is the vertical clustering of the values of the same column of several rows. For instance, the *status* of, say, 16 records could be clustered and encrypted into a single cipher. This approach resembles the PAX storage layout proposed in [1]. Vertical clustering can be applied even if there is only a single small column in a table, but it is more difficult to integrate into a query processor if the query processor is not already based on a PAX storage layout.

Indexing and Materialized Views: Indexes and materialized views can be defined using Cipherbase. These indexes and materialized views inherit the encryption scheme from the referenced data and take the restrictions for dynamic information leakage into account. For instance, the entries of a *Patient.name* index (for the schema described in Figure 2) would be encrypted using AES in ECB mode. Point lookups (i.e., equality predicates) could be processed using that index entirely in the UM without decrypting any keys because AES in ECB mode is deterministic. Thus, a hash index on the Patient.name column can run completely in the UM. However, processing range predicates using a B-Tree index on the column(e.g., *Patient.name LIKE "Smith%"*) would have to be processed using the TM. Currently we do not support indexes for columns that include knobs for runtime security (this would require the integration of oblivious RAM techniques in the storage subsystem). Query processing with indexes in Cipherbase is discussed in more detail in [2].

Setting Isolation Levels: The isolation level is an important tuning knob in any database system. This observation is also true for Cipherbase. In Cipherbase, however, it is particularly important because it can impact which operations can be carried out in the UM and which operations need to be carried out in the TM. Using serializability, for instance, the lock manager must interact with the TM in order to check predicates on encrypted data for key range locking. Using lower isolation levels such as snapshot isolation, the actions of the lock manager can be fully executed in the UM.

Statistics: In general, the presence and maintenance of statistics such as those needed for query optimization can be a source for information leakage. In the current design of Cipherbase, however, this kind of information leakage is precluded: All statistics are kept in an encrypted form at the server and query optimization which requires these statistics in cleartext is carried out at the (trusted) client machines (Figure 1).

5 Conclusion

The goal of the Cipherbase system is to help developers to protect their confidential data against curious attackers (e.g., database administrators) and achieve high performance at the same time. Just as in any other database system, the physical database design is crucial to achieve high performance. This paper discussed the most critical physical design decisions that are specific to a system like Cipherbase. Concretely, this paper discussed the performance implications of static data security (choice of encryption scheme), runtime data security (kind and amount of work that needs to be carried out by trusted hardware), and other tuning considerations such as clustering rows and columns. We are currently prototyping Cipherbase and in the process of evaluating and quantifying the trade-offs in using the different physical design options outlined in this paper. We believe that Cipherbase is particularly suited to be used as the infrastructure for a ï£¡secureï£¡ database-as-a-service where the goal is to achieve the confidentiality that is needed at the lowest possible cost.

References

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In *VLDB*, pages 169–180, 2001.

[2] A. Arasu et al. Orthogonal security with cipherbase. In *CIDR*, 2013.

[3] S. Bajaj and R. Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *SIGMOD*, 2011.

[4] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT '09*, 2009.

[5] K. Eguro and R. Venkatesan. FPGAs for trusted cloud computing. In *FPL*, 2012.

[6] C. Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3), 2010.

[7] H. Hacigumus, S. Mehrotra, and B. R. Iyer. Providing database as a service. In *ICDE*, pages 29–38, 2002.

[8] S. Hildenbrand, D. Kossmann, T. Sanamrad, C. Binning, F. Faerber, and J. Woehler. Query processing on encrypted data in the cloud. In *Technical Report No. 735, ETH Zurich*, 2011.

[9] Microsoft Corporation. SQL Azure. http://www.windowsazure.com/en-us/home/features/sql-azure/.

[10] R. A. Popa, C. M. S. Redfield, N. Zeldovich, et al. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.

Privacy and Integrity are Possible in the Untrusted Cloud

Ariel J. Feldman #, Aaron Blankstein *, Michael J. Freedman *, Edward W. Felten *
University of Pennsylvania
* *Princeton University*

Abstract

From word processing to online social networking, user-facing applications are increasingly being deployed in the cloud. These cloud services are attractive because they offer high scalability, availability, and reliability. But adopting them has so far forced users to cede control of their data to cloud providers, leaving the data vulnerable to misuse by the providers or theft by attackers. Thus, users have had to choose between trusting providers or forgoing cloud deployment's benefits entirely.

In this article, we show that it is possible to overcome this trade-off for many applications. We describe two of our recent systems, SPORC [13] and Frientegrity [12], that enable users to benefit from cloud deployment without having to trust providers for confidentiality or integrity. In both systems, the provider only observes encrypted data and cannot deviate from correct execution without detection. Moreover, for cases when the provider does misbehave, SPORC introduces a mechanism, also applicable to Frientegrity, that enables users to recover.

SPORC is a framework that enables a wide variety of collaborative applications such as collaborative text editors and shared calendars with an untrusted provider. It allows concurrent, low-latency editing of shared state, permits disconnected operation, and supports dynamic access control even in the presence of concurrency. Frientegrity extends SPORC's model to online social networking. It introduces novel mechanisms for verifying the provider's correctness and access control that scale to hundreds of friends and tens of thousands of posts while still providing the same security guarantees as SPORC. By effectively returning control of users' data to the users themselves, these systems do much to mitigate the risks of cloud deployment.

1 Introduction

From word processing and calendaring to online social networking, the applications on which end users depend are increasingly being deployed in the cloud. The appeal of cloud-based services is well known. They offer high scalability and availability along with global accessibility and often the convenience of not requiring end users to install any software other than a Web browser. Furthermore, they can enable multiple users to edit shared state concurrently, such as in real-time collaborative text editors.

But by now, it is also well understood that these benefits come at the cost of having to trust the cloud provider with the privacy users' data. The recent history of user-facing cloud services is rife with unplanned data disclosures [19], [26], [14], [38], and these services' very centralization of information makes them

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

attractive targets for attack by malicious insiders and outsiders. In addition, providers face pressure from government agencies world-wide to release information on demand, often without search warrants [17], [20], [32]. Finally and perhaps worst of all, providers themselves often have an economic incentive to voluntarily disclose data that their users thought was private. They have repeatedly weakened their privacy policies and default privacy settings to promote new services [29], [30], [31], [39], and frequently stand to gain by selling users' information to marketers.

Less recognized, however, is the extent to which users trust providers with the *integrity* of their data, and the harm that a malicious or compromised provider could do by violating it. A misbehaving provider could corrupt users' information by adding, dropping, modifying, or forging portions of it. But a malicious provider could be more insidious. For example, bloggers have claimed that Sina Weibo, a Chinese microblogging site, tried to disguise its censorship of a user's posts by hiding them from the user's followers but still showing them to the user [34]. This behavior is an example of provider *equivocation* [25], [22], in which a malicious service presents different clients with divergent views of the system state.

In sum, the emerging class of user-facing cloud services currently requires users to cede control of their data to third-party providers. Users are forced to trust the providers' promises — and perhaps legal and regulatory measures — that have so far failed to adequately safeguard users' data. Thus, users are currently faced with a dilemma: either forgo the advantages of cloud deployment or subject their data to a myriad of new threats.

1.1 Our Approach

In this article, we argue that for many applications, it is possible to overcome this strict trade-off. Rather than forcing users to depend on cloud providers' good behavior, we propose that cloud services should be redesigned so that users can benefit from cloud deployment without having to trust the providers for confidentiality or integrity. Cloud applications should be designed under the assumption that the provider may be actively malicious, and the services' privacy and security guarantees should be rooted in cryptographic keys known only to the users rather than in the providers' promises.

In our recent SPORC [13] and Frientegrity [12] works, we present frameworks for building a wide variety of end user services with untrusted providers. In both systems, the provider observes only encrypted data and cannot deviate from correct execution without detection. Moreover, for cases when the provider does misbehave, SPORC introduces a mechanism, also applicable to Frientegrity, that enables users to recover. It allows users to switch to a new provider and repair any inconsistencies that the provider's equivocation may have caused.

SPORC makes it possible to build applications from collaborative word processors and calendars to email and instant messaging systems that are robust in the face of a misbehaving provider. It allows low-latency editing of shared state, permits disconnected operation, and supports dynamic access control even in the presence of concurrency. Frientegrity extends SPORC's model to online social networking. It introduces novel mechanisms for verifying the provider's correctness and access control that scale to hundreds of friends and tens of thousands of posts while still providing the same security guarantees as SPORC. By effectively returning control of users' data to the users themselves, these systems do much to mitigate the risks of cloud deployment. Thus, they may clear the way for greater adoption of cloud applications.

Road map This article introduces our approach to untrusted, centralized cloud services.[1] Section 2 presents a set of concepts and assumptions that is common to both of our systems. It includes a description of our threat and deployment models and a discussion of *fork* consistency*, a technique for defending against server equivocation. In Sections 3 and 4, we summarize the design, implementation, and evaluation of SPORC and Frientegrity respectively. Finally, in Section 5 we conclude and highlight directions for future work.

[1]For a more complete presentation and related work, see [13], [12], [11].

2 System Model

Traditionally, the cloud provider maintains state that is shared among a set of users. When users read or modify the state, their client devices submit requests to the provider's servers on their behalf. The provider's servers handle these potentially concurrent requests, possibly updating the shared state in the process, and then return responses to the users' clients. Because the shared state is stored on the provider's servers either in plaintext form or encrypted under keys that the provider knows, the provider must be trusted.

In SPORC and Frientegrity, however, users' shared state is encrypted under keys that the provider does not know. As a result, instead of having the servers update the shared state in response to clients' requests, all of the code that handles plaintext runs on the clients. Clients submit encrypted updates, known as *operations*, to the servers, and the servers' role is mainly limited to storing the operations and assigning them a global, canonical order. The servers perform a few other tasks such as rejecting operations that are invalid or that came from unauthorized users. But because the provider is untrusted, the clients must check the servers' output to ensure that they have performed their tasks faithfully.

2.1 Why Have a Centralized Provider?

Because both of our systems limit the provider's role mainly to ordering and storing encrypted operations, one might wonder why they use a centralized provider at all. Indeed, many peer-to-peer group collaboration and social networking systems have been proposed (*e.g.*, [36], [18], [2], [37], [8]). But decentralized schemes have at least two major disadvantages. First, they leave an end user with an unenviable dilemma: either sacrifice availability, reliability, and convenience by storing her data on her own machine, or entrust her data to one of several federated providers that she probably does not know or trust any more than she would a centralized provider. Second, they are a poor fit for applications in which an online user needs a timely notification that her operation has been committed and will not later be overridden by an operation from another user who is currently offline. For example, to schedule a meeting room, an online user should be able to quickly determine whether her reservation succeeded. Yet in a decentralized system, she could not know the outcome of her request until she had heard from at least a quorum of other users' client. By contrast, a centralized provider can leverage the benefits of cloud deployment—high availability and global accessibility—to achieve timely commits.

2.2 Detecting Provider Equivocation

To prevent a malicious provider from forging or modifying clients' operations without detection, SPORC and Frientegrity clients digitally sign all their operations with their users' private keys. But as we have discussed, signatures are insufficient because a misbehaving provider could still equivocate about the history of operations.

To mitigate this threat, both systems enforce *fork* consistency* [23].[2] In fork*-consistent systems, clients share information about their individual views of the history by embedding it in every operation they send. As a result, if clients to whom the provider has equivocated ever communicate, they will discover the provider's misbehavior. The provider can still *fork* the clients into disjoint groups and only tell each client about operations by others in its group, but then it can never again show operations from one group to the members of another without risking detection. Furthermore, if clients are occasionally able to exchange views of the history out-of-band, even a provider which forks the clients will not be able to cheat for long.

[2] Fork* consistency is a weaker variant of an earlier model called *fork consistency* [25]. They differ in that under fork consistency, a pair of clients only needs to exchange one message to detect server equivocation, whereas under fork* consistency, they may need to exchange two. Our systems enforce fork* consistency because it permits a one-round protocol to submit operations, rather than two. It also ensures that a crashed client cannot prevent the system from making progress.

2.3 Threat Model

Provider We assume that a provider may be actively malicious and may attempt to equivocate or simply deny service. Because clients cannot directly prevent such misbehavior, SPORC and Frientegrity deter it by allowing clients to detect it quickly and providing them with a means to switch to a new provider and repair any inconsistencies in the system's state that the misbehavior may have caused. Both systems prevent the provider from observing the plaintext of users' shared state, and in Frientegrity users are only known to the provider by pseudonym. Nevertheless, a provider may be able to glean some information via traffic analysis and social network deanonymization techniques [1], [27]. A full mitigation of these attacks is beyond the scope of this work.

Users and clients Both systems assume that users may also be malicious and colluding with a malicious provider. As a result, users cannot impersonate other users, decrypt or modify shared state or participate in the consistency protocol unless they have been invited by an authorized user. Frientegrity extends these protections to defend against malicious authorized users. It guarantees fork* consistency as long as the number of misbehaving users with access to a given shared item is less than a predetermined constant.

2.4 Deployment Assumptions

Provider Like traditional providers, providers in our systems would likely employ many servers to support large numbers of users and distinct shared items. Both systems enable such scalability by allowing the provider to partition the systems' state into logically distinct portions that can be managed independently on different servers. In SPORC, each collaboratively-edited piece of state, called a *document*, is entirely self-contained, leading naturally to a shared-nothing architecture [35]. In Frientegrity, each element of each user's social networking profile (*e.g.*, "walls," photo albums, comment threads) can reside on a different servers, and the system minimizes the number of costly dependencies between them. We assume that all of the operations performed on a given shared item are stored and ordered by a single server. But, for reliability, the provider could perform these tasks with multiple servers in a primary/backup or even in a Byzantine fault tolerant configuration.

Users and clients We aim to make realistic assumptions about users' clients. We assume that each user may connect to the provider from multiple client devices (*e.g.*, a laptop, a tablet, and a mobile phone), and that each device has its own separate view of the history of operations. In addition, we do not assume that clients retain any state other than the user's key pair which is used to sign every operation that she creates. We do not even assume that multiple clients are ever online simultaneously, and so our protocols do not rely on consensus among users or clients for correctness.

3 SPORC

SPORC, our first system, is a *generic* collaboration service that makes it possible to build a wide variety of applications such as word processing, calendaring, and instant messaging with an untrusted service provider. It enables a set of authorized users to concurrently edit a shared *document* as well as modify its access control list in real time, all without allowing the provider to compromise the document's confidentiality or integrity. It also allows users to work offline and only later synchronize their changes.

SPORC achieves these properties through a novel combination of *fork* consistency* and *operational transformation (OT)* [9]. Whereas fork* consistency enables clients to detect provider equivocation, OT provides clients with a mechanism to resolve conflicts that result from concurrent edits to the document without having to resort to locking. Perhaps most interestingly, however, OT also allows clients that have detected a malicious fork to switch to a new provider and repair the damage that the fork may have caused. In this way, the same mechanism that allows SPORC clients to merge correct concurrent operations also enables them to recover from a malicious provider's attacks.

Operational transformation OT provides a general set of rules for synchronizing shared state between clients. In OT, the application defines the set of operations from which all modifications to the document are constructed. When clients generate new operations, they apply them locally before sending them to others. To deal with the conflicts that these optimistic updates inevitably incur, each client *transforms* the operations it receives from others before applying them to its local state. If all clients transform incoming operations appropriately, OT guarantees that they will eventually converge to a consistent, reasonable state.

Central to OT is an application-specific *transformation function* $T(\cdot)$ that allows two clients whose states have diverged by a pair of conflicting operations to return to a consistent state. $T(op_1, op_2)$ takes two operations as input and returns a pair of transformed operations (op'_1, op'_2), such that if the party that initially did op_1 now applies op'_2, and the party that did op_2 now applies op'_1, the conflict will be resolved.

For example [28], suppose Alice and Bob both begin with the same local state "ABCDE", and then Alice applies $op_1 =$ 'del 4' locally to get "ABCE", while Bob performs $op_2 =$ 'del 2' to get "ACDE". If Alice and Bob exchanged operations and executed each others' naively, then they would end up in inconsistent states (Alice would get "ACE" and Bob "ACD"). To avoid this problem, the application supplies the following transformation function that adjusts the offsets of concurrent delete operations:

$$T(\text{del }x, \text{del }y) = \begin{cases} (\text{del }x-1, \text{del }y) & \text{if} \quad x > y \\ (\text{del }x, \text{del }y-1) & \text{if} \quad x < y \\ (\text{no-op}, \text{no-op}) & \text{if} \quad x = y \end{cases}$$

Thus, after computing $T(op_1, op_2)$, Alice will apply $op'_2 =$'del 2' as before but Bob will apply $op'_1 =$ 'del 3', leaving both in the consistent state "ACE".

Given this pairwise function, clients that diverge in arbitrarily many operations can return to a consistent state by applying it repeatedly. OT works in many settings, as operations, and the transforms on them, can be tailored to each application's requirements. For a collaborative text editor, operations may contain inserts and deletes of character ranges at specific cursor offsets, whereas for a key-value store, operations may contain lists of keys to update or remove.

3.1 SPORC Design

Architecture In SPORC, each client maintains its own copy of the document. When it performs a new operation, the client applies the operation to its local copy first and only later encrypts, digitally signs, and uploads the operation to one of the provider's servers. Upon receiving an encrypted operation, the server commits it to a place in the order of operations on the document and then broadcasts it to the other authorized clients. These clients verify the operation's signature and perform a consistency check (see below) to ensure that the provider has not equivocated about the order of operations. Finally, the clients decrypt the operation and use OT to transform the incoming operation into a form that be applied to their local states. An incoming operation may need to be transformed because the recipient's state may have diverged from the sender's. Other clients' operations may have been committed since the incoming operation was sent. Moreover, the recipient may have pending operations that it has applied locally but that have not yet been committed.

To enforce fork* consistency, each client maintains a *hash chain* over the history of committed operations it has received from the provider. For a set of operations $op_1, \ldots, op_n$, the value of the hash chain up to op_i is given by $h_i = H(h_{i-1}||H(op_i))$, where $H(\cdot)$ is a cryptographic hash function and $||$ denotes concatenation. When a client with history up to op_n submits a new operation, it includes h_n in its message. On receiving the operation, another client can check whether the included h_n matches its own hash chain computation over its local history up to op_n. If they do not match, the client knows that the provider has equivocated.

Access control SPORC allows the administrators of a document to grant and revoke users' access on the fly, but implementing dynamic access control raises several challenges. First, the system must provide an in-band mechanism for distributing encryption keys and supporting key changes when users are removed. Second, it must keep track of causal dependencies between operations and access control list (ACL) changes so that, for example, operations performed after a user has been expelled are guaranteed to be inaccessible to that user. Finally, it must prevent concurrent, conflicting access control changes from corrupting the ACL.

To address these challenges, ACL changes and key distribution are handled via special operations, ModifyUserOps, that are ordered along side ordinary document operations and that are subject to the same consistency guarantees. Adding a user entails submitting a ModifyUserOp containing the symmetric document encryption key encrypted under the new user's public key while removing a user involves picking a new document key for subsequent operations and submitting a ModifyUserOp containing the new key encrypted under the public keys of the remaining users. When creating any operation, including a ModifyUserOp, a client embeds a pointer in it to the last committed operation that the client has seen. Although the client may not know exactly where the operation will end up in the total order, SPORC ensures that it will never be ordered before this pointer, thereby ensuring causal consistency. SPORC also uses these pointers to detect and prevent concurrent, potentially conflicting ACL changes.

Fork recovery In normal operation, SPORC clients are constantly creating small forks between their histories whenever they optimistically apply operations locally, and are then later resolving these forks with OT. From the clients' perspective, a malicious fork caused by an equivocating provider looks similar to these small forks: a history with a common prefix followed by divergent suffixes after the fork point. Thus, SPORC can use OT to resolve a malicious fork in a similar way. To do so, clients essentially treat the operations after the fork as if they were new operations performed locally. A pair of forked clients can switch to a new provider, upload all of their common operations prior to the fork, and then resubmit the operations after the fork as if they were new. The new provider will assign these operations a new order and the clients can then use OT to resolve any conflicts just as they would in normal operation.

3.2 SPORC Implementation & Evaluation

The SPORC framework consists of a server and a client library written in Java that handle synchronization, consistency checking, and access control automatically. The application developer need only supply a data type for operations, a transformation function, and the client-side application. Notably, because the server only handles encrypted data, it is completely generic and need not contain any application-specific code. We used our framework to implement a Web-based, real-time collaborative text editor and a causally-consistent key-value store. In both cases, we were able to create applications robust to a misbehaving provider with only a few hundred lines of code. In particular, we were able to reuse the operations and transformation function from Google Wave [16], an OT-based groupware system with a trusted server, without modification.

To evaluate SPORC's practicality, we performed several microbenchmarks on our prototype implementation on a cluster of commodity machines connected by a gigabit LAN. Our experiments demonstrated that SPORC achieves sufficiently low latency and sufficient throughput to support the user-facing collaborative application for which it was designed. Under load, latency was under 34 ms with up to 16 clients and median server throughput reached 1600 ops/sec. Notably, latency was dominated by the cost of clients' 2048-bit RSA signatures, and thus it could be improved greatly by a faster algorithm such as ESIGN [22]. [3]

4 Frientegrity

Our second system, Frientegrity, extends SPORC's confidentiality and integrity guarantees to online social networking. It supports the main features of popular social networking applications such as "walls," "news feeds," comment threads, and photos, as well as common access control mechanisms such as "friends," "friends-of-friends (FoFs)," and "followers." But as in SPORC, the provider only sees encrypted data, and

[3] Complete results can be found in the full paper [13].

clients can collaborate to detect equivocation and other misbehavior such as failing to properly enforce access control.

Frientegrity's design is shaped by social networking's unique scalability challenges. Prior systems that enforced variants of fork* consistency assumed that the number of users would be relatively small or that clients would be connected to the servers most of the time. As a result, to enforce consistency, they presumed that it would be reasonable for clients to perform work that is linear in either the number of users or the number of updates ever submitted to the system. But these assumptions do not hold in social networking applications in which users have hundreds of friends, clients connect only intermittently, and users typically are interested only in the most recent updates, not in the thousands that may have come before. In addition, many previous proposals for secure social networking systems (*e.g.*, [2], [24], [5]) required work that is linear in the number of friends, if not FoFs, to revoke a friend's access (*i.e.*, to "un-friend"). But in real social networks, users may have hundreds of friends and tens of thousands of FoFs [10]. Finally, because popular networking providers have hundreds of millions of users, any consistency and access control mechanisms must be able to function even when the system's state is sharded across many servers.

4.1 Frientegrity Design

A Frientegrity provider runs a set of servers that store *objects*, each of which corresponds to a social networking construct such as a Facebook-like "wall". Like SPORC, clients submit encrypted operations on objects and the provider orders and stores them. The provider also ensures that only authorized clients (*e.g.*, those belonging to a user's friends) can write to each object. To confirm that the provider is fulfilling these roles faithfully, clients collaborate to verify any output that they receive from the provider. Whenever a client performs a read, the provider's response must include enough information to make verification possible. It must also contain the key material that allows a user with an appropriate private key to decrypt the object being read. But because Frientegrity must be scalable, the provider's responses must be structured to allow verification and key distribution to be performed *efficiently*.

For example, if Alice fetches the latest operations from Bob's wall object, the response must allow her to cheaply verify that: (1) the provider has not equivocated about the wall's contents (*i.e.*, enforcing fork* consistency), (2) every operation was created by an authorized user, (3) the provider has not equivocated about the set of authorized users, and (4) the ACL is not outdated.

Enforcing fork* consistency Many prior systems, including SPORC, used hash chains to enforce fork* consistency. But hash chains are a poor fit for social networking applications because verifying an object like Bob's wall would require downloading the entire history of posts and performing linear work on it even though the user is probably only interested in the most recent updates. This cost is further magnified because building a "news feed" requires a user to process all of her friends' walls.

As a result, Frientegrity represents an object's history as a *history tree* [4] rather than a list. A history tree allows multiple clients, each of which may have a different subset of the history, to efficiently compare their views of it. Processing time and the size of messages exchanged are logarithmic in the history size. With a history tree, a client can embed a compact representation of its view of the history [5] in every operation it creates, and clients which subsequently read the operation can compare their views to the embedded one.

Thus, when Alice reads the tail of Bob's wall, she can compare her view of the history with the one embedded in the most recent operation, which perhaps was created by Charlie. If Alice trusts Charlie, then she only has to directly check the operations that were committed since the last operation that Charlie observed. [6] Similarly, before uploading his operation, Charlie only had to check the operations after some earlier snapshot. In this way, no single client needs to examine every operation, and yet by collaborating,

[4] A history tree [6] is a growable Merkle hash tree that has been used previously for tamper-evident logging.

[5] *i.e.*, the history tree's current root hash signed by the provider.

[6] The provider may have committed other operations after Charlie's operation was uploaded but before his operation was committed.

clients can verify an object's entire history. Moreover, we can defend against collusion between a misbehaving provider and up to f malicious users by having clients look farther back in the history until they find a point that $f + 1$ clients have vouched for.

Making access control verifiable Bob's profile may be comprised of multiple objects under a single ACL. A well-behaved provider can reject operations from unauthorized users. But because it is untrusted, it must *prove* that it enforced access control correctly on every operation it returns in response to Alice's read. Thus, Frientegrity's ACL data structure must allow the provider to construct efficiently-checkable membership proofs. The ACL must also enable authorized clients to efficiently retrieve the keys necessary to decrypt the relevant objects. Moreover, because social network ACLs may be large, ACL changes and rekeying must be efficient.

To meet these requirements, we represent ACLs with a tree-like data structure that is a novel combination of a *persistent authenticated dictionary* [7] and a *key graph* [40] in which each node is a "friend." A given user's membership proof is simply a path from the root to that user's node and requires space and verification time that is logarithmic in the number of users. In addition, each node stores its user's symmetric key, and the keys are organized so that a user who can decrypt her own node key can follow a chain of decryptions up the tree and obtain the root key which is shared among all authorized users. As a result, adding or removing a user only requires a logarithmic number of keys to be changed along the path from the user's node to the root. Notably, adding or removing a FoF still only requires logarithmic work in the the number of *friends*, not FoFs. Finally, to prevent the provider from equivocating about the history changes to the ACL itself, root hashes of successive versions of the ACL tree are stored in their own fork*-consistent *ACL history* object.

Preventing ACL rollbacks Ideally, Frientegrity would treat every operation performed on every object as a single history and enforce fork* consistency on that history. But, doing so would create extensive, and often unnecessary, dependencies between objects, thereby making it difficult to spread objects across multiple servers without resorting to expensive agreement protocols (*e.g.*, Paxos [21]). Thus, for scalability, Frientegrity orders operations and enforces fork* consistency on each object independently. Weakening consistency across objects leads to new attacks, however. For example, even without equivocating about the contents Bob's wall or his ACL, a malicious provider could still give Alice an outdated ACL in order to trick her into accepting operations from a revoked user.

To mitigate this threat, Frientegrity supports *dependencies* between objects which specify that an operation in one object *happened after* an operation in another. To establish a dependency from object A to object B, a client adds a new operation to A annotated with a compact representation of the client's current view of B. In so doing, the client forces the provider to show anyone who later reads the operation a view of B that is at least as new as the one the client observed. With this mechanism, rollback attacks on Bob's ACL can be defeated by annotating operations in Bob's wall with dependencies on his ACL. Dependencies are also useful in other applications. For example, in a Twitter-like system, every retweet could have a dependency on the tweet to which it refers. In that case, a provider wishing to suppress the original tweet would not only have to suppress all subsequent tweets from the original user (because Frientegrity enforces fork* consistency on the user's feed), the provider would also have to suppress all subsequent tweets from everyone who retweeted it.

4.2 Frientegrity Implementation & Evaluation

To evaluate Frientegrity's design, we implemented a prototype that simulates a simplified Facebook-like service. Like SPORC, we evaluated our prototype on a cluster of commodity machines connected by a gigabit LAN. We conducted a series of experiments that measured latency, throughput, and network overhead. We found that Frientegrity's method of enforcing fork* consistency outperformed a hash chain-based design by a substantial margin. Whereas Frientegrity achieved latency under 10 ms even for objects comprised of 25,000 operations, a hash chain-based implementation had latency approaching 800 ms for objects containing only 2000 operations. Frientegrity also demonstrated good scalability with large numbers of friends. The latency of modifying an ACL containing up to 1000 friends was below 25 ms. [7]

5 Conclusion & Future Work

SPORC and Frientegrity enable a wide variety of cloud services with untrusted providers by employing a two-pronged strategy. First, to protect the confidentiality of users' information, both systems ensure that providers' servers only observe encrypted data. As a result, not only do they stop the provider from misusing users' data itself, they also prevent users' data from being stolen by malicious insiders or outsiders. Second, to protect the data's integrity, both systems' give clients enough information to check the provider's behavior. Thus, clients can quickly detect any deviation from correct execution, including complex equivocation about the system state.

Nevertheless, due to the diversity of cloud applications, their use cases, and their threat models, our systems can only be considered a small part of a comprehensive mitigation of the risks of cloud deployment. Much work remains, and we discuss two directions for future work here. First, Frientegrity shows that, even within our threat model of an untrusted provider, one size does not fit all. To scale to the demands of online social networking, Frientegrity uses different data structures and a more complex client-server protocol than SPORC. Future work may attempt to extend our systems' guarantees to new applications, but might employ different mechanisms. In so doing, it may shed light on general techniques for developing efficient systems with untrusted parties. Second, although our systems support many useful features, some, such as cross-object search, automatic language translation, and contextual advertising, remain difficult to implement efficiently because the provider cannot manipulate plaintext. Finding practical ways to at least partially support these capabilities would go a long way in spurring the adoption of systems like ours. These solution may well involve algorithms that operate on encrypted data (*e.g.*, [33], [4], [3]) even if fully homomorphic encryption [15] remains impractical.

Acknowledgements

We thank Andrew Appel, Matvey Arye, Christian Cachin, Jinyuan Li, Wyatt Lloyd, Siddhartha Sen, Alexander Shraer, and Alma Whitten for their insights. We also thank the anonymous reviewers of this article and of the prior works upon which it is based. This research was supported by NSF CAREER grant CNS-0953197, an ONR Young Investigator Award, and a gift from Google.

References

[1] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore Art Thou R3579X? Anonymized social networks, hidden patterns, and structural steganography. In *Proc. WWW*, May 2007.

[2] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *Proc. SIGCOMM*, Aug. 2009.

[3] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. *CRYPTO*, pages 535–552, Aug. 2007.

[4] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. TCC*, Mar. 2006.

[5] E. D. Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of twitter. Cryptology ePrint Archive, Report 2011/640, 2011. http://eprint.iacr.org/.

[6] S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *Proc. USENIX Security*, Aug. 2009.

[7] S. A. Crosby and D. S. Wallach. Super-efficient aggregating history-independent persistent authenticated dictionaries. In *Proc. ESORICS*, Sept. 2009.

[8] Diaspora. Diaspora project. http://diasporaproject.org/. Retrieved Apr. 23, 2012.

[9] C. Ellis and S. Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, 1989.

[7] Complete results can be found in the full paper [12].

[10] Facebook, Inc. Anatomy of facebook. http://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859, Nov. 2011.
[11] A. J. Feldman. *Privacy and Integrity in the Untrusted Cloud*. PhD thesis, Princeton University, 2012.
[12] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with Frientegrity: Privacy and integrity with an untrusted provider. In *Proc. USENIX Security*, Aug. 2012.
[13] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. Sporc: Group collaboration using untrusted cloud resources. In *Proc. OSDI*, Oct. 2010.
[14] Flickr. Flickr phantom photos. http://flickr.com/help/forum/33657/, Feb. 2007.
[15] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. http://crypto.stanford.edu/craig.
[16] Google. Google Wave federation protocol. http://www.waveprotocol.org/federation. Retrieved Apr. 23, 2012.
[17] Google. Transparency report. https://www.google.com/transparencyreport/governmentrequests/userdata/. Retrieved Apr. 23, 2012.
[18] M. Handley and J. Crowcroft. Network text editor: A scalable shared text editor for MBone. In *Proc. SIGCOMM*, Oct. 1997.
[19] J. Kincaid. Google privacy blunder shares your docs without permission. *TechCrunch*, Mar. 2009.
[20] D. Kravets. Aging 'privacy' law leaves cloud e-mail open to cops. *Wired Threat Level Blog*, Oct. 2011.
[21] L. Lamport. The part-time parliament. *ACM TOCS*, 16(2):133–169, 1998.
[22] J. Li, M. N. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proc. OSDI*, Dec. 2004.
[23] J. Li and D. Mazières. Beyond one-third faulty replicas in Byzantine fault tolerant systems. In *Proc. NSDI*, Apr. 2007.
[24] M. M. Lucas and N. Borisov. flyByNight: mitigating the privacy risks of social networking. In *Proc. WPES*, Oct. 2008.
[25] D. Mazières and D. Shasha. Building secure file systems out of byzantine storage. In *Proc. PODC*, July 2002.
[26] J. P. Mello. Facebook scrambles to fix security hole exposing private pictures. *PC World*, Dec. 2011.
[27] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Proc. IEEE S & P*, May 2009.
[28] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the Jupiter collaboration system. In *Proc. UIST*, Nov. 1995.
[29] K. Opsahl. Facebook's eroding privacy policy: A timeline. *EFF Deeplinks Blog*, Apr. 2010.
[30] R. Sanghvi. Facebook blog: New tools to control your experience, Dec. 2009.
[31] E. Schonfeld. Watch out who you reply to on google buzz, you might be exposing their email address. *TechCrunch*, Feb. 2010.
[32] D. J. Solove. A taxonomy of privacy. *University of Pennsylvania Law Review*, 154(3):477–560, Jan. 2006.
[33] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. IEEE S & P*, May 2000.
[34] S. Song. Why I left Sina Weibo. http://songshinan.blog.caixin.cn/archives/22322, July 2011.
[35] M. Stonebraker. The case for shared nothing. *IEEE DEB*, 9(1):4–9, 1986.
[36] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proc. SOSP*, Dec. 1995.
[37] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Better privacy for social networks. In *Proc. CoNEXT*, Dec. 2009.
[38] U.S. Federal Trade Commission. FTC accepts final settlement with twitter for failure to safeguard personal information. http://www.ftc.gov/opa/2011/03/twitter.shtm, Mar. 2011.
[39] J. Vijayan. 36 state ags blast google's privacy policy change. *Computerworld*, Feb. 2012.
[40] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM TON*, 8(1):16–30, 1998.

My Private Google Calendar and GMail

Tahmineh Sanamrad
ETH Zurich

Daniel Widmer
ETH Zurich

Lucas Braun
ETH Zurich

Patrick Nick
ETH Zurich

Donald Kossmann
ETH Zurich

Abstract

Although Cloud Applications provide users with highly available data services, they are missing privacy as a vital non-functional requirement. In this paper we leverage modern cryptography techniques to guarantee the user's privacy while the inherent functionality and portability of the cloud application remain intact. Our approach revolves on a transparent security middleware that sits between the user and the cloud service provider on a site trusted by the user. This layer accesses the request and response messages passed between the two parties in a fine-grained manner to preserve the functionalities. We implemented the methods and provide a middleware that allows users to keep their calendar information and E-Mail in an encrypted form using Google Calendar and GMail. Furthermore, we present the results of experiments with our middleware; these experiments show that the overhead to encrypt data on top of GMail and Google Calendar is negligible.

1 Introduction

Trust and privacy play a crucial role in today's applications, as more and more people and companies decide to outsource their data and IT services. There are plenty of cloud data services and web applications that help to organize and store data for free or at a low cost. Still, for some people high availability, up to date features, light-weight interfaces, backup, low maintenance costs and portability on the latest mobile devices seem all to be overshadowed by privacy doubts and suspicions.

This paper presents our experience on building a middleware to enforce privacy on top of two popular Web applications, namely Google Calendar and GMail. The goal is to use these two services without revealing any information to an attacker who has access to the Google Cloud (e.g., a Google system administrator) or who intercepts messages from or to the Google Cloud. We use a security middleware as a transparent encryption layer on a site trusted by the client. The key component in the security middleware is a proxy server that inspects and accesses the http message body, selectively encrypts/decrypts its content in a fine-grained manner, thereby preserving the original APIs. Recently in [9] a very similar approach to this paper has been suggested. However, the proxy server is installed on the client which restricts the portability but does not require an additional SSL termination step.

The main advantage of our architecture is transparency of the complicated encryption mechanism and key management for the end user. Also, given the variety of mobile devices that embed cloud data services,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

our system assures a device-independent installation and thereby portability is guaranteed. As a proof of concept, we have developed a prototype of the calendar application for the most recent mobile devices.

To recover the missing functionality due to encryption, we compose a new scheme. The resulting scheme is a hybrid of a semantically secure encryption scheme and a keyword hashing scheme. The scheme has been analysed against certain adversary models. These models correspond to an honest but curious attacker that only by looking at the encrypted data residing on the cloud tries to infer the plaintext. The performance benchmarks show that generally the latency cost of a hybrid encryption mechanism is negligible.

Enforcing privacy and compensating the missing functionality could not have been done without a carefully planned component orchestration on the middleware. Along the way, many interesting problems have been addressed and new techniques have been devised. For example in order to add sender/receiver anonymization in the GMail and Google Calendar Invitation, an additional component namely a mail transfer agent has been added to the middleware, to offer additional privacy.

The remainder of this paper is organized as follows: In section 2 the architecture of our approach is being discussed and compared against other possible approaches. In section 3 and 4 the methods invented to tackle the privacy/functionality seesaw are discussed. Section 5 looks at the war stories that are still unresolved or need further treatments. In section 6, benchmark results will show an initial comparison between different possible encryption techniques. Finally, we conclude the paper in section 7 while discussing the related work and the future prospects of the project.

2 Proxy Architecture

In this section we will first explore different possible approaches to provide security for the end user given our case studies, Google calendar and GMail. The main idea is to replace the plain text content entered into the Google calendar's web interface with a ciphertext before submitting the request to Google. In order to accomplish this, there are two possible approaches as shown in Figure 1:

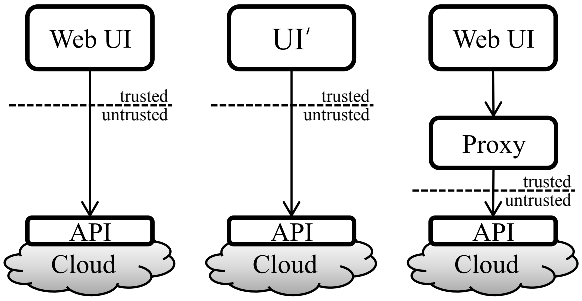
1. Having a rich client that takes care of the replacement
2. Adding a level of indirection between the client and the service provider

2.1 Rich Client vs. Proxy Middleware

As shown in Figure 1(a), currently the users directly use the API of the cloud service provider and submit their requests in plaintext. Although an SSL connection is established between the user and the cloud service provider, the data stored and processed on the cloud side is all in plaintext.

To have the data encrypted before submission, there are two possible solutions. The first solution is shown in Figure 1(b). In this approach, a new user interface is implemented on the client side. In addition to the new user interface, the security procedures are to be taken care of on the client. In our case study, Google actually provides a clean *Calendar Data API*that enables the users to access its calendar methods, such as `create()`, `edit()`, and `invite()`, while giving them a lot of flexibility to design their own desired user interfaces. A considerable amount of documentation has been written and dedicated to support developers who use this API. On the other hand, given the variety of the devices and hardware architectures, implementing such a user interface for each and every device out there is a cumbersome task with a high development cost. Additionally, the current user interfaces already provided by the cloud service providers are efficient and friendly enough. Another disadvantage is that the end user has to undergo a lot of installation and set up efforts on every single device she has.

The second approach is shown in Figure 1(c). This solution adds a level of indirection between the user and the cloud service provider. This level of indirection resides on the user's trusted site and is a proxy server that acts as an intermediary for requests from clients seeking resources from the cloud service provider. The first advantage of our approach is that the client accesses the proxy server using the same API as the one provided by the cloud service provider; this assures the transparency of the security mechanism to the end user, so the user should not even be aware of the encryption process going on behind the scenes. The second advantage is the low installation cost for the user; all that the user needs to do is to route the relevant traffic through the proxy middleware. The third advantage is the portability of our approach. This way all the user's devices can keep their current well provided and maintained interface of the cloud service provider and only route their traffic through the proxy middleware.



(a) Traditional (b) Rich Client (c) Proxy MW

Figure 1: Comparing different approaches

2.2 Make it Work

As already discussed in the previous subsection, there are several advantages associated with the proxy architecture. Having a security middleware has been previously discussed in the database encryption research area in [3], or in some works related to the Internet data storage security such as [8]. However our security middleware consists of a proxy server that catches relevant http traffic, modifies the message content in a fine-grained manner with the help of a content adaptation server, and sends it further. The components to build such a system are as follows:

- *Proxy server*. As shown in Figure 2, the client connects to the proxy server on the middleware, requesting some service from the cloud service provider. The proxy server evaluates the request according to its filtering rules. If the request is validated by the filter, the proxy provides the resource by connecting to the cloud service provider and requesting the service on behalf of the client.
- *ICAP Server*. ICAP stands for Internet Content Adaptation Protocol(RFC3507) which is used to extend transparent proxy servers, as it is shown in Figure 2. This component adapts the content of the http messages by performing the particular value added service (content encryption/decryption) for the associated client request/response.
- *Mail Transfer Agent*. The MTA mainly acts as a relay for email and operates independently from the proxy and ICAP server. It will receive messages, encrypt/decrypt the message and send it further.

As mentioned in the introduction, the privacy of the users is guaranteed through encryption. However, privacy comes in cost of functionality. Therefore, the effect of every decision in one dimension (Privacy or Functionality) must be thoroughly examined in its counteracting dimension. In the next section, we look at Google Calendar and GMail scenario separately. For each scenario we name the main functions, and what does privacy imply. We devise a hybrid encryption scheme to deal with the defined privacy requirements. Two Adversary models will be introduced to analyse the advantage of an attacker that only has access to the encrypted data. Moreover in the GMail section we show how our approach tackles the sender/receiver anonymization problem.

3 Google Calendar

In this section we focus on the Google Calendar case study. First, we state what functionality of the calendar we are interested in. Then, we describe what does privacy imply in our system. After the preliminary

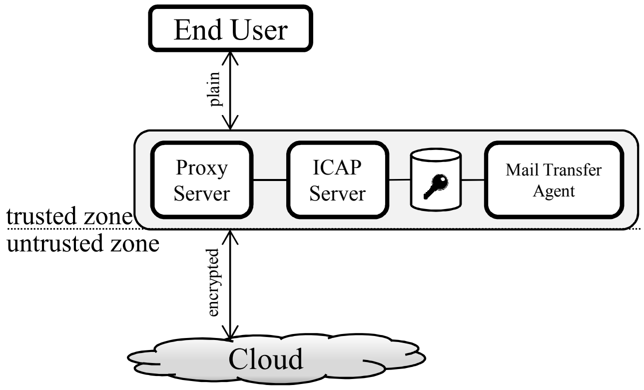


Figure 2: An overview of the architecture and components in our system

definitions, we show how the privacy is implemented and functionality is preserved.

Functionality. In the Google Calendar model the users interact with the cloud application mainly through `create()`, `display()` and `search()` functions. The users can also interact with each other through `share()` and `invite()` functions.

Privacy is defined as the inability of the adversary (e.g., an operator who has access to the Google cloud) to infer information about the events just by looking at the encrypted data. In our model with some probability the attacker will be able to guess some information, which is captured as the attacker's advantage (Section 3.2). The remainder of this section elaborates on the encryption scheme that is used to achieve this privacy and implement the Google Calendar functionality at the same time.

3.1 Hybrid Encryption Scheme

Assume we have a user Alice who wants to create and store a calendar event using an untrusted calendar application. The proxy server inspects Alice's calendar traffic (with her permission) and extracts the plain text pieces. The safest approach from this point on would be to replace the plaintext with a ciphertext generated by a semantically secure encryption scheme, such as AES operating in CBC mode [7]. Since the proxy is trusted by Alice, it will generate and store a key to be used for encryption. The randomized element of the encryption, namely the initialization vector (IV) is stored along with the ciphertext on the untrusted server to enable decryption at some point. However, if a probabilistic encryption scheme is used, one of the main calendar functions, namely search will be disabled, since the randomized element (IV) is missing to reconstruct a query that matches an entry in the cloud. Another challenge is that assume Alice has created an event called *"Bob Birthday"*, and she wants to search for *"bob"*. Even if we were using a deterministic encryption scheme, we were unable to retrieve the event because obviously Enc_K*("Bob Birthday")*$<>$ Enc_K*("bob")*. This simple example shows the need of tokenizing and normalizing the user input plaintext as well as the search query. Therefore, to guarantee both search and exact retrieval of Alice's entries, we concatenate the list of normalized keywords of the event entry to the encrypted message. However, the keywords should also not leak any information and at the same time be searchable. There are different approaches on searchable encryption[1, 2, 4, 5], but the experiments we have performed in Section 6 shows that hashing is more efficient than encryption. Thus, we devise an idealized smoothing hash function. This hash function maps a keyword to a hash value. Based on a frequency histogram of the keywords kept in the main memory of the middleware, the hash function dynamically adjusts its assignments to perfectly smooth out the frequency of the hash values produced. Based on the frequency distribution of the input keywords, the degree of collision and multi-hashed values (one value has multiple hashes) in the system will be determined. Note that in order to create an idealized smoothing hash function, we need a histogram that is built on the plaintext domain, $\mathcal{D}$. Construction 3.1.1 and figure 3 best describe our hybrid encryption scheme.

Construction 3.1.1: Let $\mathcal{SSE} = (\mathcal{IV}, \mathcal{E}nc', \mathcal{D}ec')$ be a semantically secure encryption scheme and $\mathcal{ISHS} = (\mathcal{T}ok, \mathcal{N}orm, \mathcal{H}ash, \mathcal{H}ist)$ be an idealized smoothing hash scheme on Domain, $\mathcal{D}$. We define our Hybrid scheme, $\mathcal{HS} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ to be the following:

- $\mathcal{K}$ is a random function that generates a 128-bit key for the proxy, K_p.
- $\mathcal{IV}$ is a random function that generates a 128-bit initialization vector for the each message, $iv = \mathcal{IV}()$.
- $\mathcal{E}$nc gives the plaintext message m, to the $\mathcal{E}nc'$ function of SSE to generate the ciphertext, $c = \mathcal{E}nc'(K_p, iv, m)$. The random initialization value is first concatenated to the encrypted message, then the hashed keyword list, kl, generated by $ISHS$, $kl = \mathcal{H}ash(\mathcal{N}orm(\mathcal{T}ok(m)), \mathcal{H}ist(\mathcal{D}))$ is concatenated to the encrypted message as well. The encrypted message will be $c_m = c\|iv\|kl$.
- $\mathcal{D}$ec takes the proxy key, K_p, the ciphertext and the randomized part of the encrypted message, c_m; and, by using the $\mathcal{D}ec'$ function it revives the original message $m = \mathcal{D}ec'(K_p, iv, c)$.

Although a semantically secure encryption scheme encrypts the messages, the keyword list generated by the hash function weakens the security of our hybrid scheme. In the next section we look at two adversary models that analyse the advantage of the adversary given our definition of privacy in the beginning of Section 3.

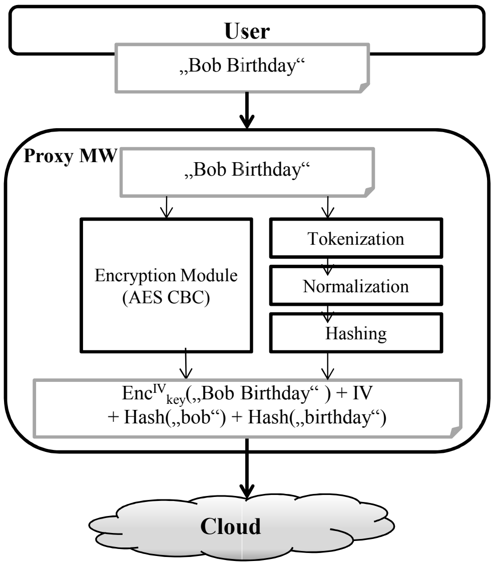


Figure 3: Hybrid Encryption Scheme

3.2 Adversary Models

In this section we introduce two security definitions, *Frequency Indistinguishability* and *Event Uncertainty*. We then analyse the advantage of the adversary in each model.

3.2.1 Frequency Indistinguishability

A deterministic scheme leaks the frequency distribution of the underlying plaintext which is not desirable. In order to show the resistance of our scheme against the frequency analysis of the keywords, we introduce a new security definition called *Frequency Indistinguishability*.

Frequency Indistinguishability: The advantage of an adversary in distinguishing pairs of ciphertexts just by looking at the frequency distribution of the messages they encrypt should be negligible. Let $\mathcal{HS}$ be our hybrid encryption scheme from Construction 3.1.1. For an adversary $\mathcal{A} = (A_1, A_2)$, we define its IND-Freq advantage as:

$$Adv_{\text{HS}}^{\text{ind-freq}}(\mathcal{A}) = Pr[Exp_{\text{HS}}^{\text{ind-freq-1}}(\mathcal{A}) = 1] - Pr[Exp_{\text{HS}}^{\text{ind-freq-0}}(\mathcal{A}) = 1]$$

For $b \in \{0, 1\}$ the experiments $Exp_{HS}^{ind-freq-b}(A)$ can be viewed in Experiment 1 and We say that HS is ind-freq secure if the ind-freq advantage of any adversary against HS is small.

Proof: The proof relies on two important procedures in the Experiment 1. First, $\mathcal{R}ebalance$, assures that M_0 and M_1 have identical histograms, i.e. the histograms are formed with the same number of buckets and the same frequency distribution. Second, the $\mathcal{S}ort$ sorts the result of our hashing scheme based on their frequency. Applying $\mathcal{S}ort$ on the hashes will map the output of $\mathcal{S}ort$ to the output of $\mathcal{R}ebalance$

in terms of frequency distribution. By definition the $\mathcal{R}ebalance$ procedure assures identical frequency distribution between the two chosen frequency distributions by adversary. Hence, we can say that deciding to which frequency distribution the output of $\mathcal{S}ort$ belongs, is not better than a random guess. Therefore, the advantage of adversary $\mathcal{A}$, in experiment 1 is negligible.

Corollary 1: Given the above model we can conclude that in order to be safe against frequency analysis on our domain, $\mathcal{D}$, we need to $\mathcal{R}ebalance$ the histogram of our domain with a uniform frequency distribution. In other words,
$hist_0 \leftarrow \mathcal{H}ist(\mathcal{D});\ hist_1 \leftarrow \mathcal{H}ist(UNIFORM); \mathcal{R}ebalance(hist_0, hist_1)$

Experiment 1 :$Exp_{\text{HS}}^{\text{ind-freq-b}}(A)$

$(\mathcal{M}_0, \mathcal{M}_1) \xleftarrow{\$} \mathcal{A}_1$
if $|\mathcal{M}_0| \neq |\mathcal{M}_1|$ **then return** $\perp$
$hist_0 \leftarrow \mathcal{H}ist(\mathcal{M}_0)$
$hist_1 \leftarrow \mathcal{H}ist(\mathcal{M}_1)$
$\mathcal{R}ebalance(hist_0, hist_1)$
let $m_1^j, m_2^j, ..., m_l^j$ be the elements of M_j for $j \in \{0, 1\}$
if $\exists i : 1 \leq i \leq l$ and $|m_i^0| \neq |m_i^1|$ **return** $\perp$
for $j = 1$ **to** l
| $\quad h_j \leftarrow \mathcal{H}ash(m_j^b, hist_b)$
| $\quad H_j^b \leftarrow h_j$
$H^b \leftarrow \mathcal{S}ort(H^b)$
$d \leftarrow \mathcal{A}_2(h_1, h_2, ..., h_l)$
return d

3.2.2 Event Uncertainty

We define another adversary model that is only applicable to calendar data. This adversary takes advantage of the repetitive pattern or length of certain event. By applying additional background knowledge, the adversary might be able to make strong guesses about certain events that the user is likely to attend. For example a yearly event is most likely to be an anniversary (e.g. birthdays) or adversary knows that the user is a professor and is most likely to attend a certain conference on certain days. In order to decrease the strength of the adversary's guesses we add noise to our system. Therefore, we introduce a new security definition called *Event Uncertainty*.

Event Uncertainty. Given a time interval what is the advantage of an adversary in guessing whether an event is real or not. Let I be the interval, I_{all} be the set of all events in I, and I_{real} be the set of real events in I. Let $e \in I_{all}$ be an event, we define a function $\tau(e)$ to return the duration of an event. Hence, the advantage of the adversary in the interval of I will be:

$$Adv^I(\mathcal{A}) = \frac{\sum_{e \in I_{real}} \tau(e)}{\sum_{e \in I_{all}} \tau(e)} \tag{3}$$

The naive way of adding noise is by random. We believe, however that there are more clever ways to add noise to the encrypted data to break repetition patterns or fuzzify the duration of certain events. For example in case of a birthday event, changing it to a monthly event would conceal its yearly pattern. Nevertheless, we have not developed a concrete model to optimally add noise to the calendar data. Creating event uncertainty could also be done by changing date and time of an event. Unfortunately, this approach is hard to implement, because of the prefetching procedure going on in the background of the calendar page.

4 GMail

In addition to Google Calendar, we studied the proxy architecture to achieve privacy on top of GMail. GMail also provides a sophisticated API and a Web interface and it involves confidential information that we would like to protect from *honest and curious* adversary that has access to the Google cloud.

Functionality. In GMail, the users interact with the cloud application mainly through `compose()`, `send()`, `search()`, and `receive()` functions. In addition, GMail provides functionality to filter spam, group conversations, and to spell-check.

Privacy is again defined as the inability of the adversary to infer information just by looking at the encrypted emails. The information we want to hide is the sender, recipient, title and message body. The message body is encrypted using the same hybrid encryption scheme discussed in Section 3.1. Concealing sender and recipient from the Google Mail Server undermines the main functionality of a mail server. In the next section we will show how to resolve this issue.

4.1 Sender and Recipient Anonymization

In this section, we suggest a sender/recipient anonymization technique that reduces GMail to be solely a storage engine and an email management interface. Our method is explained through an example. Assume Alice is a user of our proxy's mail service. Her gmail account is *alice@gmail.com*. The proxy, assigns another email address to Alice, called *alice@proxy.ethz.ch*. This email address is in fact the address which Alice can be contacted by other people. However, in order to access and operate her email account:*alice@proxy.ethz.ch* she needs to login to her gmail account, *alice@gmail.com*. Now assume Alice wants to send an email to Bob. Bob is not a proxy user, thus he cannot read encrypted contents. As shown in figure 4, Alice logs into her GMail account, composes an email with Bob's address in the recipient field and presses send. What happens in the proxy is that the email content and recipients are extracted, the content and actual recipients are encrypted and stored in the message body. Moreover, the recipients are replaced by a random recipient residing on the proxy's mail server. The message will be stored on Google servers, but now it is sent back to the proxy instead of Bob. This time the mail transfer agent on the proxy receives the message and decides what to do with the content. In this case since Bob is not a proxy user, the message will be decrypted. Bob's address is extracted from the message body and the message is sent by *alice@proxy.ethz.ch*. This approach guarantees recipient anonymization.

Now let us walk through a scenario in which a plaintext message is sent by Bob to Alice. In order for Bob to send a message to Alice, he needs to use *alice@proxy.ethz.ch* as her address. The mail server on the proxy receives the message, encrypts it, and sends it to Alice's gmail account, *alice@gmail.com*. Alice can simply access her inbox by logging in to her gmail account, and the proxy guarantees that Alice sees all her emails in plain text. This approach guarantees sender anonymization [10].

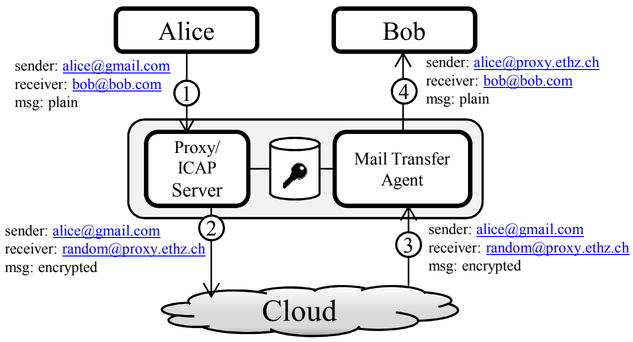


Figure 4: Sending an Email

5 War Stories

In this section we look at the challenges that are still unresolved or can be more gracefully done in our system as future work.

GMail Spam Filter. The spam filter feature of GMail inspects the sender, content and subject of an email. Encrypting emails will disable it. A possible solution is to implement a Spam-filter by adding a content-based filter to the mail server on the proxy middleware[10].

GMail Conversation Grouping. A very convenient feature of the Gmail web interface is that emails get grouped into conversations if there are many emails being sent back and forth between certain people. This feature improves the inbox organization of the user. There are two conditions for this grouping to happen: the sender/recipient addresses must match and the subject line must be equal apart from the well-known prefixes like "Re:"Our encryption scheme is designed in a probabilistic way such that two equal plaintexts will never lead to equal ciphertexts. The consequence of this is that on the servers, the first email and the replying email have different subjects; therefore, Gmail is unable to group them into a conversation. To solve this we need to use a deterministic encryption for the title and recipients. Using a deterministic encryption has its own pitfalls and is prone to frequency analysis.

SSL Interception. Google, like other secure web applications, uses SSL (Secure Socket Layer) protocol which encrypts the segments of network connections above the Transport Layer, using asymmetric cryptography for privacy and a keyed message authentication code for message reliability. Normally, a proxy server should not read the content of the message as an intermediary between the Google and end user. Nevertheless, some proxy servers offer options to decrypt SSL traffic and allow transparent SSL traffic redirection; thus, instead of having an encrypted SSL tunnel between the end user's browser and Google's server, our proxy server terminates the Google's SSL traffic at the proxy level. Sequentially, the ICAP server extracts the content to be encrypted and reconstructs the HTTP message on its way to the Google server. The adapted content is then sent to the end-user, while presenting a forged SSL certificate to the user's browser. In other words, our middleware basically performs something similar to a Man in the middle attack, but the big difference is that the user agrees to give our middleware the permission to access its contents. The user can give the permission by either confirming a certificate exception on the browser for the first time visiting the Google domain, or installing our root CA Certificate in the trusted root Certificate Authority list of the browser.

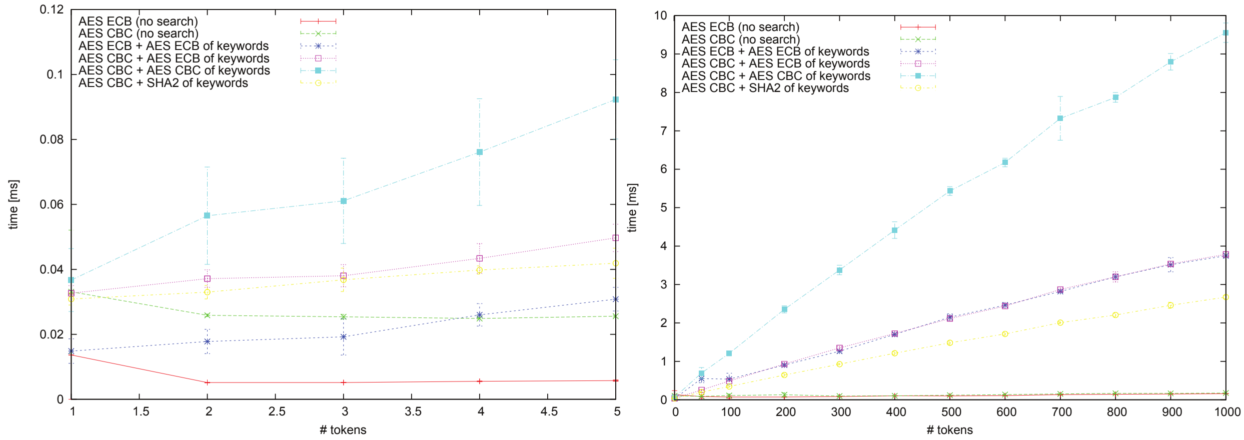
Query Log Attack To perform search and at the same time be safe against frequency attacks and have event uncertainty, we have added collision, multi-hash values and noise to our encrypted messages. Each of them have its own consequences. Adding collision, will cause the search result to retrieve more than expected, but our system easily eliminates false positives on the security middleware. Having multi-hash values, however will cause the proxy to submit a disjunctive search request, which reveals the connection between the keywords and eventually leaking the frequency distribution. Last but not least, noise added to the encrypted data will never be searched for; thus, it also leaks information about what events are fake. Solving these problems remains a future challenge.

Chosen Plaintext Attack. So far we have only analysed an honest but curious attacker, that only tries to infer the plaintext by looking at the ciphertext. A well-known adversary model that has been neglected so far, is an attacker that can use the proxy server and has access to the encrypted data on the untrusted cloud. This strong adversary is able to perform adaptive chosen plaintext attacks on the proxy. However, it can be easily stopped by assigning each user its own key and hash function, instead of using a proxy-wide one.

Key Management. Using multiple keys per user adds security, but on the other hand complicates the searchability of the calendar and again is vulnerable against query log attacks because of submitting a disjunctive search query. In case of sharing a calendar, the middleware needs to be able to deal with giving and revoking keys to and from other users.

6 Experiments

In the previous section we have shown that some level of privacy can be achieved while preserving the key functionality of the cloud application. However, security comes with a cost. In this section we will look at the encryption cost using different encryption and hashing methods.[1] The goal of these experiments is to show the cost of keyword extraction vs. no keyword extraction, and also hashing vs. encryption. As a baseline we use a deterministic encryption scheme (AES ECB) and a probabilistic encryption scheme (AES CBC) without keyword extraction. We then add the keyword extraction phase and measure the cost of hashing vs. encryption. In several previous work such as [6, 9] a symmetric encryption of the keywords has been proposed. Therefore, we have also included AES ECB + AES ECB encryption of the keywords to represent a pure deterministic encryption, AES CBC + AES ECB encryption of keywords to represent the probabilistic encryption of messages and deterministic encryption of keywords, and AES CBC + AES CBC of the keywords to represent probabilistic encryption of both messages and keywords, to cover all the schemes proposed by previous work.



(a) Encryption Time of Events

(b) Encryption Time of Documents

Figure 5: Comparison of different encryption methods

The following conclusions are to be drawn from our experiments.

Conclusion 1) By looking at figure 5 we can see that generally the latency introduced by applying security modules is considered to be small, in order of milliseconds.

Conclusion 2) In graph 5(a) we can see that in most of the cases the latency does not scale with the small number of tokens, whereas in graph 5(b), we clearly see that the latency increases linearly with the message size. This fact shows that in very small documents the cost of message encryption dominates the cost of keyword extraction and hashing (or encryption), but in bigger documents, the cost of keywords extraction overshadows the cost of the message encryption.

Conclusion 3) In graph 5(b) we can see that deterministically encrypting the keywords in ECB mode is much faster than having a probabilistic encryption scheme for the keywords.

Conclusion 4) Finally, in graph 5(b) we can see that hashing the keywords is much faster than symmetrically encrypting them in any way (ECB or CBC). The security implications introduced by hashing have been discussed in section 3.1.

Please note that these experiments are not showing the latency of our system. Given that google has unknown flow control mechanisms, performing scalability benchmarks is a challenge and is left for future work.

[1]The experiments were conducted on a Lenovo Thinkpad T400 having an Intel Core Duo cpu clocked at 2.80 GHz, 4 GB of RAM and ubuntu 12.04 as operating system.

7 Conclusion

This paper describes the system we have implemented to solve privacy issues in web applications by having a transparent encryption layer. The goal is to preserve the advantages of cloud-based web services (i.e., low cost, no administration, great user experience) without sacrificing privacy, performance, functionality and portability. The paper showed how this goal could be achieved for the Google Calendar and GMail service. A proxy architecture was devised and a number of new techniques were implemented in order to preserve the Google Calendar and GMail functionality on encrypted data. In particular, a new encryption scheme was presented that allows to search on the encrypted data. Experiments also support the fact that the proposed security scheme is more efficient among the other suggested schemes from previous works. Additionally, Performance experiments showed that the latency impact is tolerable.

There are several avenues for future research. First, we would like to apply our approach to other Web Applications such as Google Contacts and Google Docs and the services provided by other providers (e.g., Microsoft, Yahoo, Amazon, etc.). A number of new technical challenges need to be addressed to support all the features of these services, but the general approach and the proxy architecture should still be applicable.

7.1 Related Work

The proxy architecture has been also recently proposed by [9]. However, in their approach the proxy resides on the client machine; therefore limiting the portability. Also in our paper we have devised a new way to perform search on encrypted data. This topic is not new. There has been several related work in this area. In [4] they have a very similar use case, in which the user wants to search for a keyword in her encrypted emails. Their solution suggests that sender encrypts every keyword in her mail with the user's public key. Another work [1] suggests several encryption schemes to enable search on encrypted documents. In[2] they came up with secure indexes to enable search on encrypted data, but these approaches [1, 2, 4–6] are based on the fact that the untrusted server is programmable and can implement our search mechanism and data structure, whereas in our case we know almost nothing about how search is implemented on the Google servers.

Another set of related work is iDataGuard [8] which is an interoperable security middleware for untrusted Internet data storage. Their main goal is to adapt to heterogeneity of interfaces of Internet data providers and enforce security constraints. They also allow search on encrypted data using a special indexing technique. However, they search at a file-level granularity whereas we provide fine-grained encryption to preserve privacy of more complicated web application than just pure data storage.

References

[1] Dawn X. Song et al., *Practical Techniques for Searches on Encrypted Data* 2000: IEEE Symposium on Security and Privacy.

[2] Eu-Jin Goh., *Secure Indexes* 2003: IACR Cryptology ePrint Archive.

[3] Ernesto Damiani et al., *Balancing confidentiality and efficiency in untrusted relational DBMSs* 2003: CCS.

[4] Dan Boneh et al., *Public Key Encryption with keyword Search* 2004: EUROCRYPT.

[5] Yan-Cheng Chang and Michael Mitzenmacher *Privacy Preserving Keyword Searches on Remote Encrypted Data* 2005: ACNS.

[6] Reza Curtmola et al., *Searchable symmetric encryption* 2006: CCS.

[7] Jonathan Katz and Yehuda Lindell *Introduction to Modern Cryptography* 2007: Chapman & Hall/CRC.

[8] Ravi Jammalamadaka et al., *iDataGuard: an interoperable security middleware for untrusted internet data storage* 2008: USENIX.

[9] Mamadou H. Diallo et al.,*CloudProtect:Managing Data Privacy in Cloud Applications* 2012:IEEE Cloud.

[10] Patrick Nick *Encrypting Gmail* 2012: ETH Zuerich Master Thesis.

Tweeting with Hummingbird: Privacy in Large-Scale Micro-Blogging OSNs

Emiliano De Cristofaro
PARC
edc@parc.com

Claudio Soriente
ETH Zurich
claudio.soriente@inf.ethz.ch

Gene Tsudik
UC Irvine
gene.tsudik@uci.edu

Andrew Williams
UC Irvine
andrewmw@uci.edu

Abstract

In recent years, micro-blogging Online Social Networks (OSNs), such as Twitter, have taken the world by storm, now boasting over 100 million subscribers. As an unparalleled stage for an enormous audience, they offer fast and reliable diffusion of pithy tweets to great multitudes of information-hungry and always-connected followers with short attention spans. At the same time, this appealing information gathering and dissemination paradigm prompts some important privacy concerns about relationships between tweeters, followers and interests of the latter.

In this paper, we assess privacy in today's Twitter-like OSNs and describe an architecture and a trial implementation of a privacy-preserving service called Hummingbird *– a variant of Twitter that protects tweet contents, hashtags and follower interests from the (potentially) prying eyes of the central server. We argue that, although inherently limited by Twitter's mission of scalable information dissemination, the attainable degree of privacy is valuable. We demonstrate, via a working prototype, that* Hummingbird*'s additional costs are tolerably low. We also sketch out some viable enhancements that might offer even better privacy in the long term.*

1 Introduction

Online Social Networks (OSNs) offer multitudes of people a means to communicate, share interests, and update others about their current activities. Alas, as their proliferation increases, so do privacy concerns with regard to the amount and sensitivity of disseminated information. Popular OSNs (such as Facebook, Twitter and Google+) provide customizable "privacy settings", i.e., each user can specify other users (or groups) that can access her content. Information is often classified by categories, e.g., personal, text post, photo or video. For each category, the account owner can define a coarse-grained access control list (ACL). This strategy relies on the trustworthiness of OSN providers and on users appropriately controlling access to their data. Therefore, users need to trust the service not only to respect their ACLs, but also to store and manage all accumulated content.

OSN providers are generally incentivized to safeguard users' content, since doing otherwise might tarnish their reputation and/or result in legal actions. However, user agreements often include clauses that let providers mine user content, e.g., deliver targeted advertising [11] or re-sell information to third-party services. Moreover, privacy risks are exacerbated by the common practice of caching content and storing it off-line (e.g., on tape backups), even after users explicitly delete it. Thus, the threat to user privacy becomes

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

permanent. Therefore, it appears that a more effective (or at least an alternative) way of addressing privacy in OSNs is by delegating control over content to its owners, i.e., the end-users. Towards this goal, the security research community has already proposed several approaches [2, 8, 12] that allow users to explicitly authorize "friends" to access their data, while hiding content from the provider and other entities.

However, the meaning of *relationship*, or *affinity*, among users differs across OSNs. In some, it is not based on any real-life trust. For example, micro-blogging OSNs, such as Twitter and Tumblr, are based on (short) information exchanges among users who might have no common history, no mutual friends and possibly do not trust each other. In such settings, a user publishes content labeled with some "tags" that help others search and retrieve content of interest. Furthermore, privacy in micro-blogging OSNs is not limited to content. It also applies to potentially sensitive information that users (subscribers or followers) disclose by searches and interests. Specifically, tags used to label and retrieve content might leak personal habits, political views, or even health conditions. This is particularly worrisome considering that authorities are increasingly monitoring and subpoenaing social network content [6]. Therefore, we believe that privacy mechanisms for micro-blogging OSNs, such as Twitter, should be designed differently from personal affinity-based OSNs, such as Facebook.

1.1 Motivation

Twitter is clearly the most popular micro-blogging OSN today. It lets users share short messages (tweets) with their "followers" and enables enhanced content search based on keywords (referred to as *hashtags*) embedded in tweets. Over time, Twitter has become more than just a popular micro-blogging service. Its pervasiveness makes it a perfect means of reaching large numbers of people through their always-on mobile devices. Twitter is also the primary source of information for untold millions (individuals as well as various entities) who obtain their news, favorite blog posts or security announcements via simple 140-character tweets.

Users implicitly trust Twitter to store and manage their content, including: tweets, searches, and interests. Thus, Twitter possesses complex and valuable information, such as tweeter-follower relationships and hashtag frequencies. As mentioned above, this prompts privacy concerns. User interests and trends expressed by the "Follow" button represent sensitive information. For example, looking for tweets with a hashtag #TeaParty, (rather than, say, #BeerParty), might expose one's political views. A search for #HIVcure might reveal one's medical condition and could be correlated with the same user's other activity, e.g., repeated appearances (obtained from a geolocation service, such as Google Latitude) of the user's smartphone next to a clinic. Based on its enormous popularity, Twitter has clearly succeeded in its main goal of providing a ubiquitous real-time push-based information sharing platform. However, we believe that it is time to re-examine whether it is reasonable to trust Twitter to store and manage content (tweets) or search criteria, as well as to enforce user-defined ACLs.

1.2 Overview

This paper describes Hummingbird: a privacy-enhanced variant of Twitter. Hummingbird retains key features of Twitter while adding several privacy-sensitive ingredients. Its goal is two-fold:

1. Private fine-grained authorization of followers: a tweeter encrypts a tweet and chooses who can access it, e.g., by defining an ACL based on tweet content.

2. Privacy for followers: they subscribe to arbitrary hashtags without leaking their interests to any entity. That is, Alice can follow all #OccupyWS tweets from the New York Times (NYT) such that neither Twitter nor NYT learns her interests.

Hummingbird can be viewed as a system composed of several cryptographic protocols that allow users to tweet and follow others' tweets with privacy. We acknowledge, from the outset, that privacy features advocated in this paper would affect today's business model of a micro-blogging OSN. Since, in Hummingbird, the provider does not learn tweet contents, current revenue strategies (e.g., targeted advertising) would be difficult to realize. Consequently, it would be both useful and interesting to explore economic incentives for providing privacy-friendly services and not just in the context of micro-blogging OSNs. However, this topic is beyond the scope of this paper.

To demonstrate Hummingbird's practicality, we implemented it as a web site on the server side. On the user side, we implemented a Firefox extension to access the server, by making cryptographic operations transparent to the user. Hummingbird imposes minimal overhead on users and virtually no extra overhead on the server; the latter simply matches tweets to corresponding followers.

2 Privacy in Twitter

This section overviews Twitter, discusses fundamental privacy limitations and defines privacy in micro-blogging OSNs.

2.1 Twitter

As the most popular micro-blogging OSN, Twitter (http://www.twitter.com) boasts over 100 million active users worldwide, including: journalists, artists, actors, politicians, socialites, regular folks and crackpots of all types, as well as government agencies, NGOs and commercial entities [14]. Its users communicate via 140-character messages, called *tweets*, using a simple web interface. Posting messages is called *tweeting*. Users may subscribe to other users' tweets; this practice is known as *following*. Basic Twitter terminology is as follows:

- A user who posts a tweet is a *tweeter*.
- A user who follows others' tweets is a *follower*.
- The centralized entity that maintains profiles and matches tweets to followers is simply *Twitter*.

Tweets are labeled and retrieved (searched) using ***hashtags***, i.e., strings prefixed by a "#" sign. For example, a tweet: "I don't care about #privacy on #Twitter" would match any search for hashtags "#privacy" or "#Twitter". An "@" followed by a user-name is utilized for mentioning, or replying to, other users. Finally, a tweet can be re-published by other users, and shared with one's own followers, via the so-called *re-tweet* feature. Tweets are *public* by default: any registered user can see (and search) others' public tweets. These are also indexed by third party services – such as Google – and can be accessed by application developers through a dedicated streaming API. All public tweets are also posted on a public website (http://twitter.com/public_timeline), that keeps the tweeting "timeline" and shows twenty most recent messages. Tweeters can restrict availability of their tweets by making them "private" – accessible only to authorized followers [1]. Tweeters can also revoke such authorizations, using (and trusting) Twitter's *block* feature. Nonetheless, whether a tweet is public or private, Twitter collects all of them and forwards them to intended recipients. Thus, Twitter has access to all information within the system, including: tweets, hashtags, searches, and relationships between tweeters and their followers. Although this practice facilitates dissemination, availability, and mining of tweets, it also intensifies privacy concerns stemming from exposure of information.

Defining privacy in a Twitter-like system is a challenging task. Our definition revolves around the server (i.e., Twitter itself) that needs to match tweets to followers while learning as little as possible about both. This would be trivial if tweeters and followers shared secrets [15]. It becomes more difficult when they have no common secrets and do not trust each other.

2.2 Built-in Limitations

From the outset, we acknowledge that privacy attainable in Twitter-like systems is far from perfect. Ideal privacy in micro-blogging OSNs can be achieved only if no central server exists: all followers would receive all tweets and decide, in real-time, which are of interest. Clearly, this would be unscalable and impractical in many respects. Thus, a third-party server is needed. The main reason for its existence is the matching function: it binds incoming tweets to subscriptions and forwards them to corresponding followers. Although we want the server to learn no more information than an adversary observing a secure channel, the very same matching function precludes it. Similarly, the server learns whenever multiple subscriptions match the same hashtag in a tweet. Preventing this is not practical, considering that a a tweeter's single hashtag might have a very large number of followers. It appears that the only way to conceal the fact that multiple followers are interested in the same hashtag (and tweet) is for the tweeter to generate a distinct encrypted (*tweet, hashtag*) pair for each follower. This would result in a linear expansion (in the number of followers of a hashtag) for each tweet. Also, considering that all such pairs would have to be uploaded at roughly the same time, even this unscalable approach would still let the server learn that – with high probability – the same tweet is of interest to a particular set of followers.

Note that the above is distinct from server's ability to learn whether a given subscription matches the same hashtag in multiple tweets. As we discuss in [4], the server can be precluded from learning this information, while incurring a bit of extra overhead. However, it remains somewhat unclear whether the privacy gain would be worthwhile.

2.3 Privacy Goals and Security Assumptions

Our privacy goals are commensurate with aforementioned limitations.

- Server: learns minimal information beyond that obtained from performing the matching function. We allow it to learn which, and how many, subscriptions match a hashtag; even if the hashtag is cryptographically transformed. Also, it learns whether two subscriptions for the same tweeter refer to the same hashtag. Furthermore, it learns whenever two tweets by the same tweeter carry the same hashtag.
- Tweeter: learns who subscribes to its hashtags but not *which* hashtags have been subscribed to.
- Follower: learns nothing beyond its own subscriptions, i.e., it learns no information about any other subscribers or any tweets that do not match its subscriptions.

Our privacy goals, coupled with the desired features of a Twitter-like system, prompt an important assumption that the server must adhere to the **Honest-but-Curious (HbC)** adversarial model. Specifically, although the server is assumed to faithfully follow all protocol specifications, it might attempt to passively violate our privacy goals. According to our interpretation, the HbC model precludes the server from creating "phantom" users. In other words, the server does not create spurious (or fake) accounts in order to obtain subscriptions and test whether they match other followers' interests.

The justification for this assertion is as follows. Suppose that the server creates a phantom user for the purpose of violating privacy of genuine followers. The act of creation itself does not violate the HbC model. However, when a phantom user contacts s a genuine tweeter in order to obtain a subscription, a protocol transcript results. This transcript testifies to the existence of a spurious user (since the tweeter can keep a copy) and can be later used to demonstrate server misbehavior.

We view this assumption as unavoidable in any Twitter-like OSN. The server provides the most central and the most valuable service to large numbers of users. It thus has a valuable reputation to maintain and any evidence, or even suspicion, of active misbehavior (i.e., anything beyond HbC conduct) would result in a significant loss of trust and a mass exodus of users.

2.4 Definitions

We now provide some definitions to capture privacy loss that is unavoidable in Hummingbird in order to efficiently match tweets to subscriptions. Within a tweet, we distinguish between the actual message and its hashtags, i.e., keywords used to tag and identify messages.

Tweeter Privacy. An encrypted tweet that includes a hashtag ht should leak no information to any party that has not been authorized by the tweeter to follow it on ht. In other words, only those authorized to follow the tweeter on a given hashtag can decrypt the associated message. For its part, the server learns whenever multiple tweets from a given tweeter contain the same hashtag.

Follower Privacy. A request to follow a tweeter on hashtag ht should disclose no information about the hashtag to any party other than the follower. That is, a follower can subscribe to hashtags such that tweeters, the server or any other party learns nothing about follower interests. However, the server learns whenever multiple followers are subscribed to the same hashtag of a given tweeter.

Matching Privacy. The server learns nothing about the content of matched tweets.

3 Private Tweeting in Hummingbird

In this section, we present Hummingbird architecture and protocols.

3.1 Architecture

Hummingbird architecture mirrors Twitter's, involving one central server and an arbitrary number of registered users, that publish and retrieve short text-based messages. Publication and retrieval is based on a set of hashtags that are appended to the message or specified in the search criteria.

Similar to Twitter, Hummingbird involves three types of entities:

1. ***Tweeters*** post messages, each accompanied by a set of hashtags that are used by others to search for those messages. For example, Bob posts a message: "I care about #privacy" where "#privacy" is the associated hashtag.
2. ***Followers*** issue "follow requests" to any tweeter for any hashtag of interest, and, if a request is approved, receive all tweets that match their interest. For instance, Alice who wants to follow Bob's tweets with hashtag "#privacy" would receive the tweet: "I care about #privacy" and all other Bob's tweets that contain the same hashtag.
3. ***Hummingbird Server*** (HS) handles user registration and operates the Hummingbird web site. It is responsible for matching tweets with follow requests and delivering tweets of interest to users.

3.2 Design Overview

Unlike Twitter, access to tweets in Hummingbird is restricted to authorized followers, i.e., they are hidden from HS and all non-followers. Also, all follow requests are subject to approval. Whereas, in Twitter, users can decide to approve all requests automatically. Also, Hummingbird introduces the concept of ***follow-by-topic***, i.e., followers decide to follow tweeters and specify hashtags of interest. This feature is particularly geared for following high-volume tweeters, as it filters out "background noise" and avoids inundating users with large quantities of unwanted content. For example, a user might decide to follow the New York Times on #politics, thus, not receiving NYT's tweets on, e.g., #cooking, #gossip, etc. Furthermore, follow-by-topic might allow tweeters to charge followers a subscription fee, in order to access premium content. For example, Financial Times could post tweets about stock market trends with hashtag #stockMarket and only authorized followers who pay a subscription fee would receive them.

Key design elements are as follows:

1. Tweeters encrypt their tweets and hashtags.
2. Followers can *privately* follow tweeters on one or more hashtags.
3. HS can *obliviously* match tweets to follow requests.
4. Only authorized (previously subscribed) followers can decrypt tweets of interest.

At the same time, we need to minimize overhead at HS. Ideally, privacy-preserving matching should be as fast and as scalable as its non-private counterpart.

Intuition. At the core of Hummingbird architecture is a simple Oblivious PRF (OPRF) technique. Informally, an OPRF [3, 5, 7, 9, 10] is a two-party protocol between sender and receiver. It securely computes

$f_s(x)$ based on secret index s contributed by sender and input x – by receiver, such that the former learns nothing from the interaction, and the latter only learns $f_s(x)$.

Suppose Bob wants to tweet a message M with a hashtag ht. The idea is to derive an encryption key for a semantically secure cipher (e.g., AES) from $f_s(ht)$ and use it to encrypt M. (Recall that s is Bob's secret.) That is, Bob computes $k = H_1(f_s(ht))$, encrypts $Enc_k(M)$ and sends it to HS. Here, $H_1 : \{0,1\}^* \to \{0,1\}^{\tau_1}$ is a cryptographic hash function modeled as a random oracle and τ_1 is a polynomial function of the security parameter τ.

To follow Bob's tweets with ht, another user (Alice) must first engage Bob in an OPRF protocol where she plays the role of the receiver, on input ht, and Bob is the sender. As a result, Alice obtains $f_s(ht)$ and derives k that allows her to decrypt all Bob's tweets containing ht. Based on OPRF security properties, besides guaranteeing tweets' confidentiality, this protocol also prevents Bob from learning Alice's interests, i.e., he only learns that Alice is among his followers but not which hashtags are of interest to her. Alice and Bob do not run the OPRF protocol directly or in real time. Instead, they use HS as a conduit for OPRF protocol messages.

Once Alice establishes a follower relationship with Bob, HS must also efficiently and *obliviously* match Bob's tweets to Alice's interests. For this reason, we need a secure tweet labeling mechanism.

To label a tweet, Bob uses a PRF, on input of an arbitrary hashtag ht, to compute a cryptographic token t, i.e., $t = H_2(f_s(ht))$ where H_2 is another cryptographic hash function, modeled as a random oracle: $H_2 : \{0,1\}^* \to \{0,1\}^{\tau_2}$, with τ_2 polynomial function of the security parameter τ. This token is communicated to HS along with the encrypted tweet.

As discussed above, Alice must obtain $f_s(ht)$ beforehand, as a result of running an OPRF protocol with Bob. She then computes the same token t, and uploads it to HS. OPRF guarantees that t reveals no information about the corresponding hashtag. HS only learns that Alice is one of Bob's followers. From this point on, HS obviously matches Bob's tweets to Alice's interests. Upon receiving an encrypted tweet and an accompanying token from Bob, HS searches for the latter among all tokens previously deposited by Bob's followers. As a result, HS only learns that a tweet by Bob matches a follow request by Alice.

OPRF choice. Although Hummingbird does not restrict the underlying OPRF instantiation, we selected the construct based on Blind-RSA signatures (in ROM) from [5], since it offers the lowest computation and communication complexities. One side-benefit of using this particular OPRF is that it allows us to use standard RSA public key certificates. At the same time, the Hummingbird architecture can be seamlessly instantiated with any other OPRF construction.

Support for Multiple Hashtags. To ease presentation, we assumed that all tweets and follow requests contain a single hashtag, however, there is an easy extension for tweeting or issuing follow requests on multiple hashtags.

Suppose Bob wants to tweet a message M and associate it with n hashtags, $ht_1^*, \ldots, ht_n^*$: anyone with a follow request accepted on *any* of these hashtags should be able to read the message. We modify the tweeting protocol as follows: Bob selects $k^* \leftarrow_R \{0,1\}^{\tau_1}$, and computes $ct^* = Enc_{k^*}(M)$. He then computes: $\{\delta_i^* = H(ht_i^*)^{d_b} \bmod N_b\}_{i=1}^n$. Finally, Bob sends to HS $(ct^*, \{Enc_{H_1(\delta_i^*)}(k)\}_{i=1}^n, \{H_2(\delta_i^*)\}_{i=1}^n)$. The rest of the protocol involving matching at HS, as well as Alice's decryption, is straightforward, thus, we omit it.

Further, suppose Alice would like to follow Bob on *any* hashtags: $(ht_1, \ldots, ht_l)$: we only need to extend the OPRF interaction between them to l parallel executions and let Alice deposit the results to HS – the rest of the protocol is unmodified and is omitted to ease presentation.

4 System Prototype

We implemented Hummingbird as a fully functioning research prototype. It is available at http://sprout.ics.uci.edu/hummingbird. In this section, we demonstrate that: (1) by using efficient cryptographic mechanisms, Hummingbird offers a privacy-preserving Twitter-like messaging service, (2) Hummingbird introduces no overhead on the central service (HS) (thus raising no scalability concerns), and (3) Hummingbird's performance makes it suitable for real-world deployment.

4.1 Server-side

In the description below, we distinguish between server- and client-side components. Hummingbird's server-side component corresponds to HS, introduced in Section 3. It consists of three parts: (1) database, (2) JSP classes, and (3) Java back-end. We describe them below.

Database. Hummingbird employs a central database to store and access user accounts, encrypted tweets, follow requests, and profiles.

JSP Front-end. The visual component is realized through JSP pages. that allow users to seamlessly interact with a back-end engine via the web browser. Main web functionalities include: registration, login, issuing/accepting/finalizing a request to follow, tweeting, reading streaming tweets, and accessing user profiles.

Java Back-end. Hummingbird functionality is realized by a Java back-end running on HS. The back-end is deployed in Apache Tomcat. The software includes many modules; we omit their descriptions. The back-end mainly handles access to the database, populates web pages, and performs efficient matching of tweets to followers using off-the-shelf database mechanisms.

4.2 Client-side

Users interface with the system via the Hummingbird web site. We implemented each operation in Hummingbird as a web transaction. Users perform them from their own web browsers. However, several client-side cryptographic operations need to be performed outside the browser: to the best of our knowledge, there is no browser support for complex public-key operations such as those needed in OPRF computation.

To this end, we introduce, on the client-side, a small Java back-end to perform cryptographic operations. Then, we design a Firefox extension (HFE) to store users' keys and to *automatically* invoke appropriate Java code for each corresponding action. Its latest version is compatible with Firefox 3.x.x and is available from http://sprout.ics.uci.edu/hummingbird.

Client-side Java Back-end (CJB). Hummingbird users are responsible for generating their RSA keys, encrypting/decrypting tweets according to the technique presented in Section 3, and performing OPRF computations during follow request/approval. These cryptographic operations are implemented by a small Java back-end, CJB, included in the HFE presented below. CJB relies on the Java Bouncy Castle Crypto library.

Hummingbird Firefox Extension (HFE). As mentioned above, HFE is the interface between the web browser and the client-side Java back-end, included as part of the extension package. Extension code connects to it using Java LiveConnect [13]. Once installed, HFE is completely transparent to the user. HFE is used for:

Key management. At user registration, HFE automatically invokes RSA key generation code from CJB, stores (and optionally password-protects) public/private key in the extension folder, and lets browser report the public key to HS.

Following. For each of the three steps involved in requesting to follow a tweeter, the user is guided by Hummingbird web site, however, CJB code is executed to realize the corresponding cryptographic operations. This is done automatically by HFE.

Tweet. When a user tweets, HFE transparently intercepts the message with its hashtags and invokes CJB code to encrypt the message and generate appropriate cryptographic tokens.

Read. Followers receive tweets from HS that match their interests, These tweets are encrypted (recall that matching is performed obliviously at HS). HFE automatically decrypts them using CJB code and replaces web page content with the corresponding cleartext.

5 Conclusion

This paper presented one of the first efforts to assess and mitigate erosion of privacy in modern micro-blogging OSNs. We analyzed privacy issues in Twitter and designed an architecture (called Hummingbird) that offers Twitter-like service with increased privacy guarantees for tweeters and followers alike. While the degree of privacy attained is far from perfect, it is still valuable considering current total lack of privacy and some fundamental limitations inherent to the large-scale centralized gather/scatter message dissemination paradigm. We implemented Hummingbird architecture and evaluated its performance. Since almost all cryptographic operations are conducted off-line, and none is involved to match tweets to followers, resulting costs and overhead are very low. Our work clearly does not end here. In particular, several extensions, including revocation of followers, anonymity for tweeters as well as unlinking same-hashtag tweets, require further consideration and analysis.

References

[1] Twitter Privacy Policy. https://twitter.com/privacy, 2011.

[2] F. Beato, M. Kohlweiss, and K. Wouters. Scramble! your social network data. In *PETS*, 2011.

[3] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO*, 2009.

[4] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the Time of Twitter. In *S&P*, 2012.

[5] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Computational and Bandwidth Complexity. In *FC*, 2010. http://eprint.iacr.org/2009/491.

[6] K. Dozier. CIA Tracks Revolt by Tweet, Facebook. http://abcn.ws/uFdpVQ, 2011.

[7] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, 2005.

[8] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in Online Social Networks. In *WONS*, pages 49–54, 2008.

[9] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, 2009.

[10] S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. In *SCN*, 2010.

[11] H. Jones and J. Soltren. Facebook: Threats to privacy. In *Ethics and the Law on the Electronic Frontier Course*, 2005.

[12] W. Luo, Q. Xie, and U. Hengartner. FaceCloak: An Architecture for User Privacy on Social Networking Sites. In *CSE*, 2009.

[13] Mozilla Developer Network. LiveConnect. https://developer.mozilla.org/en/LiveConnect, 2011.

[14] Official Twitter Blog. One hundred million voices. http://blog.twitter.com/2011/09/one-hundred-million-voices.html, 2011.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *S&P*, 2000.

The MDM series of conferences has established itself as a prestigious forum for the exchange of innovative and significant research results in mobile data management. The term mobile in MDM has been used from the very beginning in a broad sense to encompass all aspects of mobility. The conference provides unique opportunities for researchers, engineers, practitioners, developers, and users to explore new ideas, techniques, and tools, and to exchange experiences.

We invite submissions of industrial papers, PhD forum papers, and proposals for demonstrations and seminars. MDM 2013 will host eight workshops, with call for papers open until the end of January 2013. The conference also features Tutorials and Panels.
MDM 2013 is to be hosted by Università degli Studi di Milano in its historic palaces and gardens in the very center of Milan.

We welcome you to join us at MDM 2013!
http://mdm2013.dico.unimi.it/

IMPORTANT DATES

Industrial papers submission: *December 20, 2012*

PhD forum submission: *January 9, 2013*

Demo submission: *January 23, 2013*

Conference dates: *June 3-6, 2013*

GENERAL CHAIRS
Claudio Bettini (University of Milan, Italy)
Ouri Wolfson (U. of Illinois at Chicago, USA)

PROGRAM CHAIRS
X. Sean Wang (Fudan University, China)
Cyrus Shahabi (U. of Southern California, USA)
Michael Gertz (Heidelberg Univ., Germany)

STEERING COMMITTEE LIAISON
Arkady Zaslavsky (CSIRO, Australia)

INDUSTRIAL TRACK CHAIRS
Dipanjan Chakraborty (IBM Research)
Massimo Valla (Telecom Italia)

WORKSHOP CHAIRS
Daniele Riboni (University of Milan, Italy)
Jianliang Xu (Hong Kong Baptist University)

ADVANCED SEMINAR CHAIRS
Takahiro Hara (Osaka University, Japan)
Thierry Delot (University of Valenciennes, Italy)

PANEL CHAIRS
Archan Misra (Singapore Management U.)
Juha Laurila (Nokia Research Center)

DEMO CHAIRS
Yan Huang (North Texas University, USA)
Demetris Zeinalipour (University of Cyprus)

PHD FORUM CHAIRS
Ralf Hartmut Güting (FernUniversität, Hagen)
Dario Freni (Google)

PUBLICITY CHAIR
Man Lung Yiu (Hong Kong Polytechnic U.)
Spiros Bakiras (John Jay College, SUNY)
Christos Efstratiou (University of Cambridge)

PROCEEDINGS CHAIR
Xing Xie (Microsoft Research Asia, China)

FINANCE CHAIR
Claudio Ardagna (University of Milan, Italy)

LOCAL ORGANIZATION CHAIR
Sergio Mascetti (University of Milan, Italy)

WEB CHAIR
Dragan Ahmetovic (University of Milan, Italy)

CALL FOR PARTICIPATION

29th IEEE International Conference on Data Engineering
Brisbane, Australia
April 8-12, 2013

www.icde2013.org

The annual ICDE conference addresses research issues in designing, building, managing, and evaluating advanced data-intensive systems and applications. It is a leading forum for researchers, practitioners, developers, and users to explore cutting-edge ideas and to exchange techniques, tools, and experiences.

Venue:
Brisbane is the capital of the Sunshine State, Queensland. It has a population of about 2 million, making it the third-largest city in Australia. With the Gold Coast to the south and the Sunshine Coast to the north, year-round warm climate, spectacular scenery and pleasant locals, Brisbane has lots to offer. The conference will be held at the Sofitel Hotel located within walking distance of Brisbane 's Central Business District.

Affiliated Workshops:

- ICDE Ph.D. Symposium
- Data-Driven Decision Guidance and Support Systems (DGSS)
- Data Engineering Meets the Semantic Web (DESWEB)
- Data Management and Analytics for Semi-Structured Business Processes (DMA4SP)
- Data Management in the Cloud (DMC)
- Graph Data Management: Techniques and Applications (GDM)
- Intelligent Data Processing on Health (IDP)
- Mobile Data Analytics (MoDA)
- Privacy-Preserving Data Publication and Analysis (PrivDB)
- Self-Managing Database Systems (SMDB)

Conference Events:

- Research papers
- Industrial papers
- Demos
- Keynotes
- Tutorials
- Panels
- Workshops

General Chairs:

- Rao Kotagiri — *The University of Melbourne, Australia*
- Beng Chin Ooi — *National University of Singapore, Singapore*

Program Committee Chairs:

- Christian S. Jensen — *Aarhus University, Denmark*
- Chris Jermaine — *Rice University, USA*
- Xiaofang Zhou — *The University of Queensland, Australia*

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398