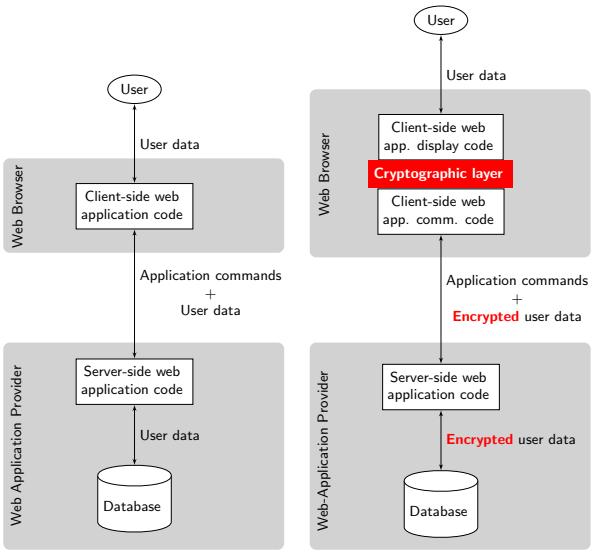
Private Use of Untrusted Web Servers via Opportunistic Encryption

Mihai Christodorescu
mihai@us.ibm.com, IBM T.J. Watson Research Center

Abstract

Users clamor for online services hosted on remote web servers. As a result, there is a growing concern about the security and privacy of the data uploaded by users to such remote services, which are under the control of potentially untrusted parties. A fair amount of work has focused on preventing, detecting, and correcting security breaches of web services, with limited efforts spent on the privacy concerns.

Here we argue that the assurance of online data privacy must go beyond legal or social contracts, which provide only *post-facto* redress, to employ technical solutions. We sketch an architecture that enhances the privacy of data sent to remote web servers, in spite of any actions by the parties controlling the web servers. Towards this end, we propose the use of client-side opportunistic encryption and decryption to allow the user to control the future use of any data they upload to a web server. We examine the challenges involved and proposed new research directions.



(a) Current web-application data flow does not allow for user control over data sent to the provider.

(b) Proposed privacy-preserving data flow uses a cryptographic layer to constrain usage of data sent to the provider.

Figure 1: In our architecture, the user's data is protected using encryption as soon as it leaves the user's computer.

1 Introduction

Users wish to use web applications[1] and at the same time preserve the privacy of their data even as it is stored online. Yet once an user submits the data to the web application, she relinquishes all control over the data to the web-application provider. The privacy of user's data can be breached by disclosure (e.g., data theft at the provider site) or by abuse (e.g., the provider using the data for purposes other than intended). The situation is compounded by the fact that the provider has partners with whom it shares the data in the normal course of business – the user would have to check the privacy policies of each provider involved. Legal solutions that expect providers to perform due diligence every time they handle data lack preemptive value, a limitation that challenges the existing legal system [12]. We need a technical solution that tackles the current web-application "ecosystem" to shield the user's data from privacy breaches at *any* web-application provider.

A simple solution to the privacy problem is to move the data (and the code for manipulating it) back onto a computer trusted by the user. This unfortunately negates the benefits brought by web-based applications. First, web applications offer the obvious benefits of availability and ease of use, as they are accessible from any location that is connected to the Internet and on any device that can run a web browser. Second, the centralization of data and code on the provider's server, similar to mainframe computing, allows for expert management of the computing platform. The tasks of maintaining and securing the application and the data are offloaded from users to the provider's professional IT personnel. The third benefit is the additional information gained by sharing and aggregating data collected from many users. Observing trends and clustering results can help users get a better view of their own data.

The solution we consider here encrypts data when it leaves the confines of the user's machine and decrypts it only when the data arrives back on the user's machine. We thus attempt to bridge the gap between the privacy

[1] For the purposes of this paper, we use the term *web application* to refer to both web-based applications and web-based services.

benefits of local data processing and the availability and functionality benefits of web-based data processing. We wish that the data stays on the local machine, the code resides on the remote web server, and all data that leaves the local machine is protected by encryption. The original problem of privacy for web-application data is now divided in our approach into two subproblems, one of transparency on the user end and one of compatibility on the provider end. The user wants to continue using her web browser to access web applications, without needing additional actions to ensure that the privacy of her data is protected. The provider wants to continue hosting the same web application, without rewriting the code and without reducing the provided functionality.

The use of encryption to protect the data stored by the web application in the provider's database poses significant challenges, as hinted above. We make the following contributions towards realizing this vision:

- An architecture that enables privacy-preserving use of untrusted web applications, while minimizing impact on already-deployed web applications (Section 2).
- A discussion of existing solutions for adapting this architecture to multiprovider (mashup-style) web applications (Section 3) and to web applications built around data sharing (Section 4).
- An agenda of open research problems to solve before an end-to-end system can be successfully deployed (Section 5).

2 Basic Architecture

We describe now the core ideas behind our architecture. To preserve secrecy of the data after it is entered by the user, a cryptographic layer encrypts the data as the client-side web application sends to the provider and decrypts when the data travels the reverse route. Figure 1 illustrates this approach.

The cryptographic layer could be located anywhere on the data path from the user to the provider's database of Figure 1(a), though from a security perspective we have to constrain it within the bounds of web browser. Encrypting data as soon as it is entered by the user (for example, by encrypting the data fields in the browser before the client-side web application accesses them) is not desirable because many web applications perform client-side processing and local display updates that require the data in cleartext. To accommodate such client-side processing, we partition the web-application code into a display-related component and a network-related component. The cryptographic layer mediates communication between the two, transforming data appropriately.

Performing partitioning of the client-side code poses similar challenges to those solved by the Privtrans project [3]. The display component, handling privileged data (i.e., cleartext data), is the trusted code, while the untrusted code is the network-communication component. Moving the cryptographic layer further away from user input means that this layer has to determine which data to encrypt and decrypt, especially as user data and application commands are combined into a single message for the server. This is an open problem, not yet solved by existing techniques.

As described, the client-side code partitions are separated only at the logical level, without any mechanism enforcing the separation. We expect that the partitioning tool ensures isolation between the three client-side components (display, cryptography, and networking) in a manner similar to inline reference monitors [11]. In our architecture, networking code must not be able to access data manipulated by the display code without mediation through the cryptographic layer. If the partitioning tool cannot guarantee such isolation, we have to rely to secure component models such as SMash to enforce the separation at runtime [4].

User-side usability. The user should not need to perform any additional work in order to have this cryptographic layer operate properly in the context of a web application. One benefit of web applications is their on-demand "installation" on the client machine, benefit that we do not wish to discard. Thus, when the user accesses a web application through her web browser, the client-side code must be automatically and transparently partitioned by the browser (or by a browser plugin) such that the cryptographic layer can be inserted. The engineering challenge is in performing this partitioning irrespective of the web application's language, be it Javascript and AJAX, Java, Flash, or any combination thereof.

The second step of importance in making this privacy architecture as transparent to the user as possible is generating the encryption key. The key must not be available to the web-application provider, so it cannot be stored online, and must be available across user devices, so the web application can be accessed from any web-enabled platform. We propose to generate a key uniquely from the password that authenticates the user to the web application and from the domain name of the provider hosting the web application. This scheme is similar to that of PwdHash project [10], where they used a Pseudo Random Function (PRF) [6] to compute a key K from the user password:

$$K = \mathrm{PRF}_{password}(dom)$$

for a given web application's domain name dom. Furthermore, we change the web-application password to be the domain name encrypted under the generated encryption key:

$$web\text{-}app\ password = E_K(dom)$$

The computed password might need to be transformed to satisfy the password requirements imposed by the web application. We refer to the PwdHash work for a possible approach. If the user needs to change their password, the data stored at the provider has to be reencrypted under the new key. The only case when this is apparently not possible is when the private data is read-only after

submission. Designing efficient and usable mechanisms to support password changes is part of future work.

As a result, the user maintains his original credentials for authenticating to the application and is not required to enter any additional information to decrypt her data, while the provider does not learn the key used to protect user data.

Provider-side functionality. A successful implementation of our architecture not only protects the privacy of user's data, but must also have no impact on the programming model used by the web-application provider. The encrypted data is transmitted to the provider's web server, where it is typically stored in a database. The goal of securing data sent to a remote web server appears similar to that of securing remote untrusted storage [5]. But the problem is complicated by the data manipulations performed by the web application. A remote storage server usually supports read, write, and access-control operations. Thus data sent to such a storage server can be safely encrypted knowing that the server will treat it as a binary blob, without any semantics attached to it. In contrast, the web application not only reads and writes the data, but also needs to compare it, perform searches over it, use it to compute averages, etc. In other words, each data item sent to the provider is typed.

The need to encrypt typed data such that operations over the plaintext have corresponding operations over the ciphertext significantly restricts the choice of encryption schemes. Depending on the type (and the associated operations) of a data value, it is possible to identify an appropriate cryptosystem, for example, RSA is a homomorphic encryption scheme for multiplication. Other cryptosystems (e.g., Benaloh's[2], Paillier's [8]) are also homomorphic, but the supported operations differ between ciphertext and plaintext (e.g., multiplication of ciphertexts is equivalent to addition of plaintexts). Such schemes would require changes in the provider's software, which is undesirable. Designing cryptosystems appropriate to various data types is an ongoing research challenge, with progress made for particular data types [7]. As an short-term goal, encryption of personally identifiable information (e.g., social-security and credit-card numbers) would address the concerns of abuse in the current web environment.

Even when the needed cryptosystems are available, the question of identifying the types of data items sent to the web-application provider remains. Currently, the data is not typed and no information is available about the operations performed on the data on the provider's server. One option is to infer types based on the observed input–output relation of the web application. A more precise option is for the web application to provide type information for the data that travels between the user's web browser and the provider's server.

We argue here that correct type information is in the provider's interest. If the type is incorrect, the web application will malfunction. If the type is too restricted, the web browser would alert the user that the corresponding data value cannot be encrypted and will be exposed to the provider and any third parties. In contrast with the current situation, our architecture makes the privacy-protecting capabilities of a web application explicit to the user. Note that simply adding type information to data does not leak private information, as the types are already known by the web application.

3 Concurrent Secure Use (aka Mashups)

Mashups are a way of composing web applications inside the web browser. We need to ensure that mashups still work within our framework. A mashup is characterized by two or more data sources accessed by a single web application. These data sources can be part of different domains (e.g., Google Maps at maps.google.com and Yahoo! Flickr at www.flickr.com), both accessed by the same client-side web-application code. In this setting, our design decision of placing the cryptographic layer as close as possible to the network-communication components allows the mashup to operate properly. Data will be decrypted as soon as it arrives to the client machine and the cleartext values can be processed, combined, and otherwise manipulated from within the client-side web-application code.

It is worth noticing that our architecture does not defend against existing cross-site attacks such as cross-site scripting and cross-site request forgery. Because encryption and decryption are done using keys tied only to the domain name of the provider hosting the data, the origin of the code making the request is not important. Thus data stored at domain A can be retrieved by web-application code originating at domains A, B, or C. In other words, cross-site scripting is still a threat and has to be prevented through other means, for example by taint tracking through combined static and dynamic program analysis [13].

4 Transitive Secure Use (aka Sharing)

Web applications often offer data sharing and collaboration capabilities. In such scenarios, the data uploaded by one user is made available to other users at the implicit or explicit instruction of the data owner. Since in this case all the users receive the same encrypted data from the provider's server, the encryption key needs to be made available to all the users involved. We distinguish here between two possible cases. First, users can share their data with a well-defined set of collaborators (e.g., one can share their online calendar with their family and friends). Then group key distribution schemes can be deployed [1], as well as simpler mechanisms based on PGP-style web of trust or on PKI infrastructures.

Second, users might wish to share their data with any number of users in the system (e.g., "everyone" can access the data). Because the set of users allowed to retrieve the data is not well defined and can change over time without the data owner's knowledge, it is not possible to use group key distribution schemes, which usually require known principals or a trusted third party. Indeed, the problem seems ill-defined in such a case, as we wish

that data is accessible to everyone *but* the web-application provider. Fundamentally, the problem is that the group sharing access to user data and the group authorized to access the web application need to be one and the same, but the group authorized to access the web application is defined by the web-application provider. One design option is to put web-application authentication in the hands of one or more trusted third parties. Then the user could use attribute-based encryption, which allows building of ciphertexts readable by anyone holding a particular set of attributes [9], to allow access to any user authenticated by particular trusted third parties.

5 Open Problems

We presented an architecture for ensuring the privacy of user data inside web applications, with minimal requirements on existing infrastructure or the current programming model. Significant challenges remain, in no small part due to the need to preserve the user's experience and the provider's programming effort. We highlight here several research problems:

- A formal type system for data sent to the web-application provider.

  The data types employed by the server-side components of a web application should be documented by the provider. Program-analysis tools are necessary to analyze server code and automatically generate type specifications. On the client, a mechanism to verify that these types are precise would increase trust in the web application.

- Cryptographic schemes preserving various data types.

  Homomorphic encryption preserves modular multiplication. Building encryption schemes for diverse types and their operations is crucial in attaining transparent end-to-end data privacy.

- Privilege separation of client-side web application code.

  This program transformation and its supporting program analysis must be fast and precise, without requiring any user interaction.

- Key distribution schemes for various access-control models.

  Current web applications provide limited flexibility in sharing (i.e., not shared, shared with friends, shared with everyone). As user demands and expertise grow, the access mechanisms will become richer and require better key distribution schemes to support richer privacy controls.

Finally, we draw attention to data integrity, a topic closely related to the privacy of data in web applications. Our architecture is geared towards protecting the confidentiality of user data from an abusive or insecure web-application provider. It does not guarantee that the data retrieved now is indeed the data submitted earlier. A research direction thus arises from the need of protecting the integrity of the data, even when split across multiple input fields and when present in multiple versions.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM conference on Computer and communications security (CCS'98)*, pages 17–26, New York, NY, USA, 1998. ACM.

[2] J. Benaloh. Dense probabilistic encryption. In *Workshop on Selected Areas of Cryptography*, pages 120–128, May 1994.

[3] D. Brumley and D. Song. Privtrans: Automatically partitioning programs for privilege separation. In *Proceedings of the 13th USENIX Security Symposium (Security'04)*, pages 57–72, 2004.

[4] F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama. SMash: Secure component model for cross-domain mashups on unmodified browsers. In *Proceedings of the 17th International World Wide Web Conference (WWW'08)*, Beijing, China, 2008.

[5] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS'03)*, pages 131–145, 2003.

[6] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[7] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD'02)*, pages 216–227, New York, NY, USA, 2002. ACM.

[8] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'99)*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 1999.

[9] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure attribute-based systems. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 99–112, New York, NY, USA, 2006. ACM.

[10] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[11] F. B. Schneider. Enforceable security policies. 3(1):30 – 50, Feb. 2002.

[12] L. Tung. Judge on privacy: Computer code trumps the law. Published by CNET News.com online at http://www.news.com/Judge-on-privacy-Computer-code-trumps-the-law/2100-1029_3-6231713.html (last accessed on Mar. 4, 2008), Feb. 22, 2008.

[13] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2007.