Research Statement

Frank Wang
MIT CSAIL
frankw@mit.edu

Web services like Google, Facebook, and Dropbox are now an essential part of people's lives. In order to provide value to users, these services collect, store, and analyze large amounts of their users' sensitive data. However, once the user provides her information to the web service, she *loses control* over how the application manipulates that data. For example, a user cannot control where the application forwards her data. Even if the service wanted to allow users to define access controls, it is unclear how these access controls should be expressed and enforced. Not only is it difficult to develop these secure access control mechanisms, but it is also difficult to ensure these mechanisms are *practical*. My research addresses these concerns. More specifically, it focuses on *building practical, secure mechanisms for protecting user data in large-scale, distributed web services.*

Broadly speaking, my research lies at the intersection of systems and cryptography. For many of my projects, I have worked closely with cryptographers to ensure my systems have *provable* security. One example of this collaboration is my Splinter system [1]. Currently, web services provide value by running user-specified queries on data, but these queries reveal information about the querier. Splinter leverages a recent cryptographic primitive, function secret sharing, to practically execute these queries without revealing sensitive information to the servers. Sieve [2] and Riverbed [3] are other examples of how my research improves the security of large-scale distributed systems. Sieve provides cryptographically enforced access control for user data stored in untrusted clouds. Riverbed provides practical information flow control for distributed systems without requiring developers to label state or write code in special languages.

In addition to collaborating within different groups in academia, I frequently seek out collaborations with industry to help me identify high-impact research problems. For example, I have worked with Keybase to understand the challenges they face with using cryptography in practice. Moreover, I have also discussed Sieve with Google and Imprivata to understand the steps needed to deploy the security mechanisms into real systems. My conversations with Rapid7 and the Twitter security team helped motivate the Splinter system. I have found these partnerships to be very helpful, and I am excited to continue them as a professor.

1 Cryptographic Protections for User Data

The amount of sensitive user data contained in datacenters is increasing rapidly. For example, hospitals regularly store electronic medical records, and services like Google and Facebook track a longitudinal behavioral profile for each user. The presence of this sensitive user information in datacenters makes the servers a lucrative target for hackers. Moreover, datacenter administrators themselves have access to the sensitive information and may unintentionally leak it, or intentionally tamper with it. Part of my research has described how to use cryptography in systems to limit server access to cleartext user data and enforce *provable security*. Compared to previous systems that execute private queries and provide cryptographic access control, Splinter and Sieve are more practical and provide more functionality.

Splinter keeps users' queries private and scales to realistic applications. Many online services let users query public datasets such as maps, flight prices, or restaurant reviews. Unfortunately, the queries reveal highly sensitive information that may compromise users' privacy. For example, travel sites can raise prices on frequently searched flights.

In Splinter, a user splits her query into multiple parts and sends each part to a different provider that holds a copy of the data. As long as any one of the providers is honest and does not collude with the others, the providers cannot determine the query contents. Splinter extends a recent cryptographic primitive called Function Secret Sharing (FSS); Splinter's modifications to FSS make Splinter up to *an order of magnitude* more efficient than prior systems which used other cryptographic techniques like

Private Information Retrieval and garbled circuits. Splinter leverages FSS to issue semantically rich queries, such as MAX and TOPK queries. Using AES-NI instructions and multicores, Splinter improves the performance of FSS by up to 2.5x. We ported several realistic applications to Splinter, including a Yelp clone and a flight search application; Splinter achieves end-to-end response latencies of less than 1.6 seconds while hiding queries from the application.

Sieve provides cryptographically enforced access control on user data. Modern web services rob users of low-level control over cloud storage—a user's single logical data set is scattered across multiple storage silos whose access controls are set by web services, not users. The consequence is that users lack control over how their data is shared with other web services.

Sieve [2] is a new platform which selectively (and securely) exposes user data to web services. Sieve has a user-centric storage model: each user uploads encrypted data to a single cloud store, and by default, only the user knows the decryption keys. Given this storage model, Sieve defines an infrastructure to carefully expose the user's data to rich, legacy web applications.

Sieve uses a variety of cryptographic techniques to enable provable security guarantees. Using attribute-based encryption, Sieve allows users to define understandable access policies that are cryptographically enforceable. Using key homomorphism, Sieve can re-encrypt user data on storage providers in situ, revoking decryption keys from web services without revealing new keys to the storage provider. Using secret sharing and two-factor authentication, Sieve protects cryptographic secrets against the loss of user devices. The result is that users can enjoy rich, legacy web applications, while benefiting from cryptographically strong controls over which data a web service can access.

2 Constraining Access to User Data

Users send their information to web services and subsequently receive valuable content, such as personalized recommendations. However, sensitive user information can leak both on the server and client. More specifically, once a user shares data with a service, how does the server securely share this data (and its derivatives) with other services? Riverbed addresses this question, and unlike previous systems, it is more practical and works with unmodified software. Moreover, on the client, if a service sends data to the user, how can we ensure that the data is minimally exposed to unauthorized client-side software? Veil provides a solution by being the first system to allow web page developers to enforce private browsing semantics with browser support.

Riverbed enforces user-defined privacy constraints in distributed web services. When users provide their information to web services, they lose control over where their data is stored, how it is computed upon, and how the data (and its derivatives) are shared with other services. Riverbed has two goals. First, Riverbed helps service operators to run applications that adhere to user-defined privacy policies; second, Riverbed helps users to verify the intent of service operators to respect user policies.

In Riverbed, server-side code is written in a managed language like Python or Java. Riverbed executes that code in a modified runtime that enforces information flow control. The runtime taints incoming data with a user-defined human-understandable privacy policy (e.g., "x.com may combine my data with the data of other users, but the results may only be sent over the network to other x.com servers"), and places data with equivalent privacy policies into an isolated instance of the full web service. This novel mechanism, called *universe isolation*, makes reasoning about privacy easier, and lowers the difficulty for server-side developers to write privacy-respecting code.

Before a user's Riverbed proxy uploads data, the proxy forces the server to remotely attest that the server executes the Riverbed software stack to assure that the server-side code will respect user-provided data policies. Traditional attestation protocols are insufficiently expressive to capture important contextual relationships between attestor-side software components; thus, we developed Cobweb [4], a new attestation system, to express behavior like nondeterministic load orders for attestor-side programs.

Unlike prior approaches that involve information flow control, Riverbed works with unmodified, legacy software, and does not require developers to reason about security lattices, or manually label code. It imposes modest performance overheads, with worst-case slowdowns of 10% for realistic applications.

Veil allows web page developers to enforce stronger private browsing semantics without browser support. All popular web browsers offer a private browsing mode. After a private session terminates, the browser is supposed to remove client-side evidence that the session occurred. Unfortunately, browsers still leak information through the file system, the browser cache, the DNS cache, and on-disk reflections of RAM such as the swap file.

Veil [5] is a new deployment framework that allows web developers to prevent these information leaks, or at least reduce their likelihood. Veil leverages the fact that, even though developers do not control the client-side browser implementation, developers do control 1) the content that is sent to those browsers, and 2) the servers which deliver that content. The Veil compiler rewrites HTML, JavaScript, and CSS in a web page, largely automating the process of eliminating a page's leakage vectors. Therefore, rewritten Veil pages only store encrypted content in the browser cache, and only expose encrypted URLs to system interfaces, making those URLs unintelligible to client-side, post-session attackers who do not possess the user's key. Veil web sites collectively store their content on Veil's *blinding servers* instead of on individual, site-specific servers, and when a user accesses a Veil page, the blinding servers and the application's JavaScript code exchange encrypted data protected by the user's key.

Veil's pages load on unmodified commodity browsers, allowing developers to provide stronger semantics for private browsing without forcing users to install or reconfigure their machines. Veil provides these guarantees even if the user does not visit a page using a browser's native privacy mode; indeed, Veil's protections are stronger than what the browser alone can provide.

3 Future Directions

Practical systems for secure multi-party computation and private queries Splinter leveraged function secret sharing (FSS) to reduce the overhead associated with private queries compared to previous work. However, open research problems still remain. For example, can private queries scale to billions or trillions of records? Furthermore, can private queries support more expressive operators like JOINs or NOT conditions? I am currently working with Shafi Goldwasser and Vinod Vaikuntanathan on exploring these questions, and plan to collaborate with other cryptographers.

Multi-party computation (MPC) allows participants to evaluate a common function without revealing their respective inputs. As hospitals, governments, and corporations increasingly transition from paper records to electronic ones, MPC-style computations represent a natural way to securely enable cross-organization data sharing. MPC is a well-studied cryptographic solution to this problem, but its impracticality has prevented widespread adoption. I believe that recent advances in garbled circuits [6, 7], together with careful design of MPC queries, can dramatically reduce the overhead of MPC systems.

Privilege-separated web services The OpenSSH and OKWS [8] papers demonstrated how privilege separation can be manually performed on relatively small, single-server applications. However, how can we help developers to implement this type of privilege separation in complex distributed systems? Reasoning about the effects of privilege separation in a large-scale application is hard, particularly if the application uses the microservice architecture—a large number of small components does not necessarily imply that the breach of a single component will not cascade throughout the system.

Models similar to this have already garnered much interest given the recent introduction of Google's BeyondCorp [9], which eliminates secure network perimeters in corporations in favor of fine-grained and tiered access privileges based on the characteristics of a user and her host machine. By isolating components with similar access control policies and using cryptography to mediate interprocess communication, a system could support privilege-separated applications.

4 Concluding Remarks

As web services accumulate increasing amounts of sensitive user data, practical security mechanisms become even more critical. I am excited to continue tackling these challenges, and I believe that my collaborations both within academia (between the systems and cryptography community) and with industry provide me with a unique perspective to solve these types of research problems.

References

[1] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical Private Queries on Public Data. In *Networked Systems Design and Implementation (NSDI)*, pages 299–313, 2017.

[2] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds. In *Networked Systems Design and Implementation (NSDI)*, pages 611–626, 2016.

[3] Frank Wang, Ronny Ko, and James Mickens. Riverbed: Enforcing User-defined Privacy Constraints in Distributed Web Services. *In submission.*

[4] Frank Wang, Yuna Joung, and James Mickens. Cobweb: Practical Remote Attestation using Contextual Graphs. In *Workshop on System Software for Trusted Execution (SysTEX)*, 2017.

[5] Frank Wang, James Mickens, and Nickolai Zeldovich. Veil: Private Browsing Semantics Without Browser-side Assistance. *To appear in Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2018.

[6] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 411–428. IEEE, 2015.

[7] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 478–492. IEEE, 2013.

[8] Maxwell N Krohn. Building secure high-performance web services with okws. In *USENIX Annual Technical Conference, General Track*, pages 185–198, 2004.

[9] Rory Ward and Betsy Beyer. Beyondcorp: A new approach to enterprise security. *;login*, 39:5–11, 2014.