

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

PROPOSAL FOR THESIS RESEARCH IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

TITLE: Protecting User Data in Large-Scale Web Services

SUBMITTED BY: Frank Wang
32 Vassar Street, #32-G978
Cambridge, MA 02139

(SIGNATURE OF AUTHOR)

DATE OF SUBMISSION: April 25, 2018
EXPECTED DATE OF COMPLETION: July 2018
LABORATORY: Computer Science and Artificial Intelligence Laboratory

BRIEF STATEMENT OF THE PROBLEM:

Web services like Google, Facebook, and Dropbox are now an essential part of peoples lives. Users willingly provide their data to these services because these services deliver substantial value in return through their centralization and analysis of data, such product recommendations and ability to easily share information. To provide this value, 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 thesis addresses these concerns.

1 Introduction

Web services like Google, Facebook, and Dropbox are now an essential part of peoples lives. Users willingly provide their data to these services because these services deliver substantial value in return through their centralization and analysis of data, such product recommendations and ability to easily share information. For example, users are willing to share their data with Facebook to learn about the social lives of their friends as well as share their own social lives more easily. Similarly, users are willing to provide their data to Amazon to discover better product recommendations. 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*. This thesis addresses these concerns. More specifically, it focuses on *building practical, secure mechanisms for protecting user data in large-scale, distributed web services*.

In this thesis, I will describe three systems that address a variety of concerns around data leakage in web applications. The first two systems focus on protecting user data against server-side leakage. Splinter leverages a recent cryptographic primitive, function secret sharing, to practically execute these queries without revealing sensitive information to the servers. The next system, Riverbed, provides practical information flow control for distributed systems without requiring developers to label state or write code in special languages. The final system, Veil, focuses on client-side leakage. Veil allows web page developers to enforce stronger private browsing semantics without browser support.

2 Splinter

2.1 Motivation

Many online services let users query large public datasets: some examples include restaurant sites, product catalogs, stock quotes, and searching for directions on maps. In these services, any user can query the data, and the datasets themselves are not sensitive. However, web services can infer a great deal of identifiable and sensitive user information from these queries, such as her current location, political affiliation, sexual orientation, income, etc. [1, 2]. Web services can use this information maliciously and put users at risk to practices such as discriminatory pricing [3, 4, 5]. For example, online stores have charged users different prices based on location [6], and travel sites have also increased prices for certain frequently searched flights [7]. Even when the services are honest, server compromise and subpoenas can leak the sensitive user information on these services [8, 9, 10].

This paper presents Splinter, a system that protects users' queries on public datasets while achieving practical performance for many current web applications. In Splinter, the user divides each query into shares and sends them to different *providers*, which are services hosting a copy of the dataset (Figure 1). As long as any one of the providers is honest and does not collude with the others, the providers cannot discover sensitive information in the query. However, given responses from all the providers, the user can compute the answer to her query.

Previous private query systems have generally not achieved practical performance because they use expensive cryptographic primitives and protocols. For example, systems based on Private Information Retrieval (PIR) [11, 12, 13] require many round trips and high bandwidth for complex queries, while systems based on garbled circuits [14, 15, 16] have a high computational cost. These approaches are especially costly for mobile clients on high-latency networks.

Instead, Splinter uses and extends a recent cryptographic primitive called Function Secret

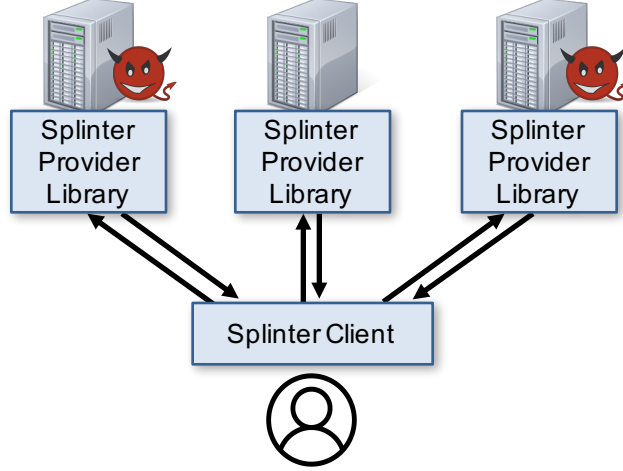


Figure 1: Splinter architecture. The Splinter client splits each user query into shares and sends them to multiple providers. It then combines their results to obtain the final answer. The user’s query remains private as long as any one provider is honest.

Sharing (FSS) [17, 18], which makes it up to an order of magnitude faster than prior systems. FSS allows the client to split certain functions into shares that keep parameters of the function hidden unless all the providers collude. With judicious use of FSS, many queries can be answered at low CPU and bandwidth cost in only a single network round trip.

Splinter makes two contributions over previous work on FSS. First, prior work has only demonstrated efficient FSS protocols for point and interval functions with additive aggregates such as SUMs [17]. We present protocols that support a more complex set of non-additive aggregates such as MAX/MIN and TOPK at low computational and communication cost. Together, these protocols let Splinter support a subset of SQL that can capture many popular online applications.

Second, we develop an optimized implementation of FSS for modern hardware that leverages AES-NI [19] instructions and multicore CPUs. For example, using the one-way compression functions that utilize modern AES instruction sets, our implementation is $2.5\times$ faster per core than a naïve implementation of FSS. Together, these optimizations let Splinter query datasets with millions of records at sub-second latency on a single server.

We evaluate Splinter by implementing three applications over it: a restaurant review site similar to Yelp, airline ticket search, and map routing. For all of our applications, Splinter

can execute queries in less than 1.6 seconds, at a cost of less than 0.02¢ in server resources on Amazon EC2. Splinter’s low cost means that providers could profitably run a Splinter-based service similar to OpenStreetMap routing [20], an open-source maps service, while only charging users a few dollars per month.

Splinter aims to protect sensitive information in users’ queries from providers. This section provides an overview of Splinter’s architecture, security goals, and threat model.

2.2 Splinter Overview

There are two main principals in Splinter: the *user* and the *providers*. Each provider hosts a copy of the data. Providers can retrieve this data from a public repository or mirror site. For example, OpenStreetMap [20] publishes publicly available map, point-of-interest, and traffic data. For a given user query, all the providers have to run it on the same view of the data. Maintaining data consistency from mirror sites is beyond the scope of this paper, but standard techniques can be used [21, 22].

As shown in Figure 1, to issue a query in Splinter, a user splits her query into *shares*, using the Splinter client, and submits each share to a different provider. The user can select any providers of her choice that host the dataset. The providers use their shares to execute the user’s query over the cleartext public data, using the Splinter provider library. As long as one provider is *honest* (does not collude with others), the user’s sensitive information in the original query remains private. When the user receives the responses from the providers, she combines them to obtain the final answer to her original query.

2.3 Security Goals

The goal of Splinter is to hide sensitive parameters in a user’s query. Specifically, Splinter lets users run *parametrized queries*, where both the parameters and query results are hidden from providers. For example, consider the following query, which finds the 10 cheapest flights between a source and destination:

```
SELECT TOP 10 flightid FROM flights
```

```
WHERE source = ? AND dest = ?  
  
ORDER BY price
```

Splinter hides the information represented by the question marks, i.e., the source and destination in this example. The column names being selected and filtered are not hidden. Finally, Splinter also hides the query’s results—otherwise, these might be used to infer the source and destination. Splinter supports a subset of the SQL language.

The easiest way to achieve this property would be for users to download the whole database and run the queries locally. However, this requires substantial bandwidth and computation for the user. Moreover, many datasets change constantly, e.g., to include traffic information or new product reviews. It would be impractical for the user to continuously download these updates. Therefore, our performance objective is to minimize computation and communication costs. For a database of n records, Splinter only requires $O(n \log n)$ computation at the providers and $O(\log n)$ communication.

2.4 Threat Model

Splinter keeps the parameters in the user’s query hidden as long as at least one of the user-chosen providers does not collude with others. Splinter also assumes these providers are *honest but curious*: a provider can observe the interactions between itself and the client, but Splinter does not protect against providers returning incorrect results or maliciously modifying the dataset.

We assume that the user communicates with each provider through a secure channel (e.g., using SSL), and that the user’s Splinter client is uncompromised. Our cryptographic assumptions are standard. We only assume the existence of one-way functions in our two-provider implementation. In our implementation for multiple providers, the security of Paillier encryption [23] is also assumed.

3 Riverbed

3.1 Motivation

In a web service, a client like a desktop browser or a smartphone app interacts with datacenter machines. Although smartphones and web browsers provide rich platforms for computation, the core application state typically resides in cloud storage. This state accrues much of its value from server-side computations that involve no participation (or explicit consent) from end-user devices.

By running the bulk of an application within VMs in a commodity cloud, developers receive two benefits. First, developers shift the burden of server administration to professional datacenter operators. Second, developers gain access to scale-out resources that vastly exceed those that are available to a single user device. Scale-out storage allows developers to co-locate large amounts of data from multiple users; scale-out computation allows developers to process the co-located data for the benefit of users (e.g., by providing tailored search results) and the benefit of the application (e.g., by providing targeted advertising).

A Loss of User Control: Unfortunately, there is a disadvantage to migrating application code and user data from a user’s local machine to a remote datacenter server: the user loses control over where her data is stored, how it is computed upon, and how the data (and its derivatives) are shared with other services. Users are increasingly aware of the risks associated with unauthorized data leakage [24, 25, 26], and some governments have begun to mandate that online services provide users with more control over how their data is used. For example, in 2016, the EU passed the General Data Protection Regulation [27]. Articles 6, 7, and 8 of the GDPR state that users must give consent for their data to be accessed. Article 15 establishes a “right to access,” mandating that users have the ability to know how their data is being processed by a company. Article 17 defines a user’s right to request her data to be deleted; Article 32 requires a company to implement “appropriate” security measures for data-handling pipelines. Unfortunately, requirements like these *lack strong*

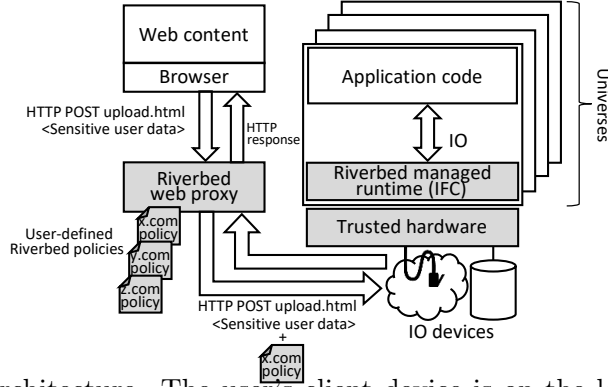


Figure 2: Riverbed’s architecture. The user’s client device is on the left, and the web service is on the right. Unmodified components are white; modified or new components are grey.

definitions and enforcement mechanisms at the systems level. Laws like GDPR provide little technical guidance to a developer who wants to comply with the laws while still providing the sophisticated applications that users enjoy.

The research community has proposed information flow control (IFC) as a way to constrain how sensitive data spreads throughout a complex system [28, 29]. IFC assigns labels to program variables or OS-level resources like processes and pipes; given a partial ordering which defines the desired security relationships between labels, an IFC system can enforce rich properties involving data secrecy and integrity. Unfortunately, traditional IFC is too burdensome to use in modern, large-scale web services. Creating and maintaining a partial ordering of labels is too difficult—the average programmer or end-user struggles to reason about data safety using the abstraction of fine-grained label hierarchies. As a result, no popular, large-scale web service uses IFC to restrict how sensitive data is processed and shared.

3.2 Our Solution: Riverbed

In this paper, we introduce Riverbed, a distributed web platform for safeguarding the privacy of user data. Riverbed provides benefits to both web developers and to end users. To web developers, Riverbed provides a *practical* IFC system which allows developers to easily “bolt on” stronger security policies for complex applications written in standard managed languages. To end users, Riverbed provides a straightforward mechanism to verify that

server-side code is running within a privacy-preserving environment.

Figure 2 describes Riverbed’s architecture. For each Riverbed web service, a user defines an information flow policy using simple, human-understandable constraints like “do not save my data to persistent storage” or “my data may only be sent over the network to `foo.com`.” In the common case, users employ predefined, templated policy files that are designed by user advocacy groups like the EFF. When the user generates an HTTP request, a web proxy on the user’s device transparently adds the appropriate data flow policy as a special HTTP header.

Within a datacenter, Riverbed leverages the fact that many services run atop managed runtimes like Python, .NET, or the JVM. Riverbed modifies such a runtime to automatically taint incoming HTTP data with the associated user policies. As the application derives new data from tainted bytes, the runtime ensures that the new data is also marked as tainted. The runtime ensures that, whenever an application tries to externalize data via the disk or the network, the externalization is only allowed if it is permitted by user policies. An application process which attempts a disallowed externalization is automatically terminated.

In Riverbed, application code (i.e., the code which the managed runtime executes) is totally unaware that IFC is occurring. Indeed, application developers have no way to read, write, create, or destroy taints and data flow policies. The advantage of this scheme is that it makes Riverbed compatible with code that has not been explicitly annotated with traditional IFC labels. However, different end users will likely define incompatible data flow policies. As a result, policy-agnostic code would quickly generate a policy violation for some subset of users; Riverbed would then terminate the application. To avoid this problem, Riverbed spawns multiple, lightweight copies of the service, one for each set of users who share the same data flow policies. We call each copy a *universe*. Since users in the same universe allow the same types of data manipulations, any policy violations indicate true problems with the application (e.g., the application tried to send sensitive data to a server that was not whitelisted by the inhabitants of the universe). Developers can then fix these violations

to ensure that code respects the desired privacy protections for the target user demographic.

Before a user’s Riverbed proxy sends data to a server, the proxy employs remote software attestation [30] to verify that the server is running an IFC-enforcing Riverbed runtime. Importantly, a trusted server will perform *next-hop attestation*—the server will not transmit user data to another network endpoint unless that endpoint is an attested Riverbed runtime whose TLS certificate name is explicitly whitelisted by the user’s data flow policy. In this manner, Riverbed enables controlled data sharing between machines that belong to different services.

3.3 Our Contributions

To the best of our knowledge, Riverbed is the first distributed IFC framework which is practical enough to support large-scale, feature-rich web services that are written in general-purpose managed languages. Riverbed preserves the traditional advantages of cloud-based applications, allowing developers to offload administrative tasks and leverage scale-out resources. However, Riverbed’s universe mechanism, coupled with a simple policy language, provides users with understandable, enforceable abstractions for controlling how datacenters manipulate sensitive data. Riverbed makes it easier for developers to comply with laws like GDPR—users give explicit consent for data access via Riverbed policies, with server-side universes constraining how user data may be processed, and where its derivatives can be stored.

We have ported several non-trivial applications to Riverbed, and written data flow policies for those applications. Our experiments shows that Riverbed enforces data flow policies with worst-case end-to-end overheads of 10%; Riverbed also supports legacy code with little or no developer intervention, making it easy for well-intentioned (but average-skill) developers to write services that respect user privacy.

3.4 Threat Model

Riverbed assumes that developers want to enforce user-defined privacy policies, but are loathe to refactor code to do so. Thus, Riverbed assumes that server-side code is weakly adversarial: poorly-designed applications may unintentionally try to leak data via explicit flows, but developers will not intentionally write code that attempts to surreptitiously leak data, e.g., via implicit flows, or via targeted attacks on the taint-tracking managed runtime.

A datacenter operator has physical access to servers, which enables direct manipulation of server RAM. So, our current Riverbed prototype assumes that datacenter operators are trusted. To ease this assumption, Riverbed could leverage a hardware-enforced isolation mechanism like Intel SGX [31]. SGX allows a secure computation to execute on a private core, with the hardware automatically encrypting and hashing the content of memory writes so that other cores are unable to inspect or modify the RAM that belongs to the secure computation. Ryoan [32] creates a distributed sandbox by representing an application as a directed graph of SGX computations that run on different servers. Unfortunately, SGX places limits on the memory size of secure applications. SGX also requires the applications to run in ring 3, forcing the code to rely on an untrusted OS in ring 0 to perform IO; the result is a large number of context switches for applications that perform many IOs [33]. Riverbed strives to be compatible with complex applications that often perform IO. So, Riverbed eschews mechanisms like SGX, and must be content to not protect against actively-malicious datacenter operators. To implement remote attestation, Riverbed does rely on trusted server-side TPM hardware, but TPMs do not protect against physical attacks on the main server hardware.

Riverbed assumes that the entire client-side is trusted, with the exception of the web content in a particular page. Buggy or malicious content may try to disclose too much information to a server. However, Riverbed ensures that whatever data is sent will be properly tagged. Since Riverbed uses TLS to authenticate network endpoints, the HTTPS certificate infrastructure must be trusted.

On the server, Riverbed’s TCB consists of the taint-tracking managed runtime, a reverse proxy that forwards requests to the appropriate universes, the TPM hardware [34] that provides the root of trust for attestation, and the daemon which servers use to attest to clients. We make standard cryptographic assumptions about the strength of the ciphers and hash functions used by the attestation protocol. Between the trusted hardware and the managed runtime are a boot loader, an OS, and possibly other systems software. Each end-user can select a different set of systems software to trust. These selections reside within the user’s policies, so that Riverbed’s client-side proxy can refuse to disclose data to untrusted server-side systems code.

4 Proposed Timeline

References

- [1] Arvind Narayanan and Vitaly Shmatikov. Myths and fallacies of personally identifiable information. *Communications of the ACM*, 53(6):24–26, 2010.
- [2] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, pages 111–125, Oakland, CA, May 2008.
- [3] Adam Tanner. Different customers, Different prices, Thanks to big data, March 21 2014. <http://www.forbes.com/sites/adamtanner/2014/03/26/different-customers-different-prices-thanks-to-big-data/>.
- [4] Felix Salmon. Why the Internet is Perfect for Price Discrimination, September 3 2013. <http://blogs.reuters.com/felix-salmon/2013/09/03/why-the-internet-is-perfect-for-price-discrimination/>.
- [5] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites. In *Proceedings of the Conference on Internet Measurement Conference (IMC)*, pages 305–318, 2014.

- [6] Jeremy Singer-Vine Jennifer Valentino-Devries and Ashkan Soltani. Websites Vary Prices, Deals Based on Users' Information, December 24 2012. Wall Street Journal.
- [7] Rick Seaney. Do Cookies Really Raise Airfares?, April 30 2013. <http://www.usatoday.com/story/travel/columnist/seaney/2013/04/30/airfare-expert-do-cookies-really-raise-airfares/2121981/>.
- [8] Ramprasad Ravichandran, Michael Benisch, Patrick Gage Kelley, and Norman M Sadeh. Capturing social networking privacy preferences. In *Proceedings of the 9th Privacy Enhancing Technologies Symposium*, pages 1–18, Seattle, WA, August 2009.
- [9] Jason Kincaid. Another Security Hole Found on Yelp, Facebook Data Once Again Put at Risk, May 11 2010. <http://techcrunch.com/2010/05/11/another-security-hole-found-on-yelp-facebook-data-once-again-put-at-risk/>.
- [10] Fahmida Y. Rashid. Twitter Breached, Attackers Stole 250,000 User Data, February 2 2013. <http://securitywatch.pcmag.com/none/307708-twitter-breached-attackers-stole-250-000-user-data>.
- [11] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of the 10th Privacy Enhancing Technologies Symposium*, pages 75–92, Berlin, Germany, 2010.
- [12] Benny Chor, Niv Gilboa, and Moni Naor. *Private information retrieval by keywords*. Citeseer, 1997.
- [13] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. *University of Waterloo, Tech. Rep. CACR*, 34:2007, 2007.
- [14] David J Wu, Joe Zimmerman, J  r  my Planul, and John C Mitchell. Privacy-preserving shortest path computation. In *Proceedings of the 2016 Annual Network and Distributed System Security Symposium*, San Diego, CA, February 2016.

- [15] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *Proceedings of the 13th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 255–273, Santa Clara, CA, March 2016.
- [16] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A system for secure multi-party computation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, pages 257–266, Alexandria, VA, October 2008.
- [17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 337–367. Sofia, Bulgaria, April 2015.
- [18] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Proceedings of the 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 640–658. Copenhagen, Denmark, May 2014.
- [19] Jeffrey Rott. Intel advanced encryption standard instructions (AES-NI). Technical report, Technical report, Intel, 2010.
- [20] OpenStreetMap. OpenStreetMap. <https://www.openstreetmap.org/>.
- [21] Renu Tewari, Thirumale Niranjan, and Srikanth Ramamurthy. WCDP: A protocol for web cache consistency. In *Proceedings of the 7th Web Caching Workshop*. Citeseer, 2002.
- [22] Chi-Hung Chi, Choon-Keng Chua, and Weihong Song. A novel ownership scheme to maintain web content consistency. In *International Conference on Grid and Pervasive Computing*, pages 352–363. Springer, 2008.
- [23] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 18th Annual International Conference on the Theory and Appli-*

- cations of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, Prague, Czech Republic, May 1999.
- [24] A. Booth. Charities hit with fines for sharing donors’ data without consent, December 7, 2016. Sophos Naked Security Blog. <https://nakedsecurity.sophos.com/2016/12/07/charities-hit-with-fines-for-sharing-donors-data-without-consent/>.
 - [25] P. Sayer. German consumer groups sue WhatsApp over privacy policy changes, January 30, 2017. PCWorld. <http://www.pcworld.com/article/3163027/private-cloud/german-consumer-groups-sue-whatsapp-over-privacy-policy-changes.html>.
 - [26] K. Zetter. Hackers Finally Post Stolen Ashley Madison Data, August 18, 2015. Wired. <https://www.wired.com/2015/08/happened-hackers-posted-stolen-ashley-madison-data/>.
 - [27] EU Parliament. GDPR Portal, 2017. <http://www.eugdpr.org/eugdpr.org.html>.
 - [28] D. Hedin and A. Sabelfeld. A Perspective on Information-Flow Control. In *Proceedings of the 2011 Marktoberdorf Summer School*, August 2001.
 - [29] P. Li, Y. Mao, and S. Zdancewic. Information Integrity Policies. In *Proceedings of the Workshop on Formal Aspects in Security and Trust*, September 2003.
 - [30] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O’Hanlon, J. Ramsdell, A. Segall, and B. Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, June 2011.
 - [31] V. Costan and S. Devadas. Intel SGX Explained, February 20, 2017. Cryptology ePrint Archive: Version 20170221:054353. <https://eprint.iacr.org/2016/086.pdf>.
 - [32] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proceedings of OSDI*, pages 533–549, 2016.

- [33] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of OSDI*, pages 267–283, 2014.
- [34] Trusted Computing Group. Trusted Platform Module (TPM) Summary. <https://trustedcomputinggroup.org/trusted-platform-module-tpm-summary/>.