Federated Code Auditing and Delivery for MPC

Frederick Jansen, Kinan Dak Albab, Andrei Lapets, and Mayank Varia

Boston University, Boston MA 02215, USA,
{fjansen, babman, lapets, varia}@bu.edu

Abstract. Secure multi-party computation (MPC) is a cryptographic primitive that enables several parties to compute jointly over their collective private data sets. MPC's objective is to federate trust over several computing entities such that a large threshold (e.g., a majority) must collude before sensitive or private input data can be breached. Over the past decade, several general and special-purpose software frameworks have been developed that provide data contributors with control over deciding whom to trust to perform the calculation and (separately) to receive the output. However, one crucial component remains centralized within all existing MPC frameworks: the distribution of the MPC software application itself. For desktop applications, trust in the code must be determined once at download time. For web-based JavaScript applications subject to trust on every use, all data contributors across several invocations of MPC must maintain centralized trust in a single code delivery service. In this work, we design and implement a federated code delivery mechanism for web-based MPC such that data contributors only execute code that has been accredited by several trusted auditors (the contributor aborts if consensus is not reached). Our client-side Chrome browser extension is independent of any MPC scheme and has a trusted computing base of fewer than 100 lines of code.

Keywords: secure multi-party computation, web security, content delivery

1 Introduction

Secure multi-party computation (MPC) permits several entities to learn joint information about their sensitive data. It has been studied for over 30 years [12, 19, 20], with several libraries and packages developed over the past decade that bring secure computing to clients on the web [6, 13] and the desktop [3, 5, 7, 9, 10, 14]. It has been deployed for social good in areas like pay equity [6], tax fraud detection [7], marketplace auctions [8], and many others.

The objective of MPC is not so much to eliminate the need to trust any particular entity, but rather to *federate* trust across several computing parties. However, all existing frameworks centralize one crucial operation: delivering the code that performs MPC itself. Note that confidentiality of the data protected by MPC relies upon the integrity of this code.

Our Contributions. In this work, we design and develop a workflow to federate delivery of MPC software.[1] We focus on web-based MPC deployments [6], where audited software is of utmost importance due to the trust-on-every-use nature of JavaScript delivery; however, we stress that our ideas apply equally well to the trust-on-first-use nature of downloaded desktop software. Within our system, data contributors rely upon the help of several services [15] who (1) deliver and (2) audit MPC software. Contributors must obtain consensus from the auditors they entrust before executing any code that will operate on their sensitive data.

Within our system, trust in the veracity of MPC software is scoped down to two sources. First, data contributors must choose trustworthy code auditors; to reduce the impact of misplaced trust, these decisions can be revoked easily at any time. Second, contributors must rely upon our Chrome extension to execute the consensus or majority vote protocol properly; to ease validation and inspire confidence, the extension is open-source[2] and designed with a small codebase.

2 Related Work

A variety of questions have been raised about whether it is prudent at all to rely on cryptographic functionalities or features implemented within web applications (e.g., using popular web languages and frameworks). One common concern focuses on the distinction between applications that require "trust on first use" vs. applications hosted on the web that require "trust on every use" [4, 18]. However, contemporary platforms and environments (including both desktop and mobile) exhibit many of the characteristics attributed to applications delivered over the web (e.g., frequent and automatic updates to the application, libraries, and even the underlying operating system). Thus, this may no longer be the most important measure of the amount of trust invested into an application.

The challenge of ensuring or validating the authenticity and integrity of code and scripts delivered to (and executed by) users can be addressed by a variety of distinct and complementary techniques and technologies. Subresource integrity (SRI) [2] involves validating web application assets served by a third party such as a content delivery network. Variants of this approach have existed for almost two decades (e.g., Netscape supported a technique for signing inline JavaScript scripts [1]). These are complementary to our proposed technique, allowing an application to ensure that imported third-party assets have not been modified inappropriately. However, SRI stops the chain of trust at the point of the web server (i.e., an attacker, whether in the form of a hacker, malicious hosting provider, or even law enforcement, could compromise a server and simply replace the code that is delivered to the end user). Thus, both the source as well as the SRI hash can be modified without anyone noticing. Our solution aims to move the trust from a single server to a much smaller signed bootstrapping

[1] While the scenario that motivates this work involves delivery of MPC software, the technique we present can be used for delivery of any web application.

[2] The source code for the implemented Chrome browser extension is available online at `https://github.com/multiparty/secure-code-delivery-extension`.

extension and a set of auditors providing the hash of the correct code. Code signing [17] and, more generally, digital signature schemes serve the complementary but distinct purpose of confirming the author of the delivered application and that the application has not been modified after being signed. However, these techniques complicate scenarios that involve application versioning and a need for delivery of the most fresh version. They also require yet another PKI, and are not supported natively by browsers for the purpose of signing and verifying the delivered code. Recent proposals include a cloud-based *secure data exchange* marketplace [11]. This is similar to our own vision of an ecosystem of modular functionalities that can be federated and delivered by incentivized entities [15] (discussed in more detail in Section 3), though in our view such functionalities (including the one presented in this work) can exist outside of a cloud setting.

3 The Secure Multi-Party Computation Ecosystem

MPC is an interactive protocol involving several participants who are connected via a networking medium that supports secure point-to-point links. We describe below several distinct *roles* [15] for MPC participants. We stress that the roles are often composable; that is, one entity can inhabit multiple roles if desired.

1. Several data *contributors* who supply the sensitive data to be analyzed.
2. An *analyst* who specifies the calculation to perform on the input data.
3. One or more *recipients* who receive the result of computing the analytic.
4. A *compute service* that provides the computational resources and network connectivity to compute the analytic in a privacy-preserving manner.
5. A code *delivery service* that provides the software necessary for contributors to encode and upload their data to the compute parties.
6. A code *auditing service* that attests to the authenticity of the code to perform secure computation. The confidentiality of the contributors' data *and* the computation's integrity rely on the trustworthiness of the delivered code.

Most existing MPC frameworks explicitly instantiate the first four roles and provide some configurability over their choices. Moreover, because contributors actively choose whether to participate in an instance of MPC, they effectively have control over which analysts, recipients, and compute services to trust. However, MPC applications to date use a single (i.e., centralized) service that delivers the JavaScript code in a web-based MPC system [6] or the source code or packaged binary in a desktop-based MPC system; the responsibility and effort of auditing the software often implicitly falls on the data contributors themselves.

We envision two possible workflows to expose and federate this trust. First, one can have several different *audited delivery services* that each perform the auditing and code delivery roles. Second, one can simply have a single delivery service and several auditors who supply a hash of the code that they have validated; this method retains federation for confidentiality and integrity but centralizes availability (i.e., the single delivery service is easier to DoS). Either way, we stress that contributors need *not* put their faith in the same set of auditors; instead, each contributor should only use the auditing services she trusts.

4 Implementation

We implemented a signed Google Chrome extension that (1) allows the application to be hosted on an untrusted server and (2) allows verification of the JavaScript code by multiple auditors (with a majority vote deciding whether to execute the code). The lightweight extension has fewer than 100 lines of code and requires only two permissions (defined in advance): sending requests to external servers and accessing the current open tab. It provides a pop-up panel attached to a button in Chrome's toolbar (a.k.a. a *browser action*) into which the user can enter the application URL. The auditing/delivery services' URLs are either encoded in the original URL or pre-defined inside the extension. The extension fetches the application code as well as the SHA-256 hash over SSL/TLS from the auditors' delivery services. The hashes are subsequently compared with the hash of the page's source code. If a majority of the auditors provide a matching hash, the extension loads the code into the browser and executes it. It also displays a table with the delivery service URL and hash match status for each auditor. The threshold required to trust the script (e.g., majority or consensus) can be passed to the extension along with the URLs of the auditors' code delivery services.

The extension does not allow the website to recover gracefully if the number of hashes that match the code's hash does not meet the threshold. Instead, the extension shows an error message resembling what a user sees when visiting a website with an invalid SSL certificate. Users who do not install the extension can use the application URL directly in the browser to load a fully functioning web page. It is up to the application's developer to decide whether this is sensible. One option is to show a warning indicating the risk of not using the extension.

5 Discussion and Future Work

Installing an extension is not ideal and limits widespread adoption. Ideally, browsers would provide this functionality natively, just as mobile and desktop OSs encourage or require applications to be signed. Also, if JavaScript code execution requires multiple auditors and a consensus vote, website updates require a synchronized effort by all parties involved. This is burdensome for web development, as updates can and do occur often. For our system to work smoothly, a new continuous deployment workflow is needed that seamlessly pushes the web application to auditors, lets them reach consensus on the new code, publishes the hashes, and simultaneously updates the server copy. This is not an urgent problem for our MPC use cases [6], as code updates coincide with deployments that occur no more than a few times per year. For applications with more frequent updates, the choice can be made to let the extension recover gracefully from a mismatched hash (letting the user decide whether to use the application).

While our motivation for this work is verified MPC code delivery, the technique fits *any* web scenario that requires authenticated code execution. One example is Coindash: clients lost $7M during the initial coin offering when hackers changed the website's source code, replacing the wallet address with their own [16]. This could have been avoided if a solution such as ours was in place.

References

1. Signing Software with Netscape Signing Tool 1.1. https://docs.oracle.com/cd/E19957-01/816-6169-10/contents.htm, [Accessed: July 13, 2017]
2. Subresource Integrity. https://www.w3.org/TR/SRI/, [Accessed: July 13, 2017]
3. VIFF. http://viff.dk/, [Accessed: June 20, 2017]
4. Arcieri, T.: Whats wrong with in-browser cryptography? https://tonyarcieri.com/whats-wrong-with-webcrypto, [Accessed: July 11, 2017]
5. Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: CCS. pp. 257–266. ACM (2008)
6. Bestavros, A., Lapets, A., Varia, M.: User-centric distributed solutions for privacy-preserving analytics. Communications of the ACM 60(2), 37–39 (2017)
7. Bogdanov, D., Jõemets, M., Siim, S., Vaht, M.: How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In: Financial Crypto. pp. 227–234. Springer (2015)
8. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T.P., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Financial Crypto. LNCS, vol. 5628, pp. 325–343. Springer (2009)
9. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.: Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In: Usenix Security Symposium. Usenix (2010)
10. Ejgenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: SCAPI: The secure computation application programming interface. Cryptology ePrint Archive 2012/629
11. Gilad-Bachrach, R., Laine, K., Lauter, K., Rindal, P., Rosulek, M.: Secure data exchange: A marketplace in the cloud. Tech. rep. (June 2016)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Proc. of the 19th Annual ACM Symposium on Theory of Computing. pp. 218–229. ACM (1987)
13. Jarrous, A., Pinkas, B.: Canon-mpc, a system for casual non-interactive secure multi-party computation using native client. In: Proc. of the 12th ACM Workshop on Privacy in the Electronic Society. pp. 155–166. ACM (2013)
14. Keller, M., Scholl, P., Smart, N.P.: An architecture for practical actively secure mpc with dishonest majority. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. pp. 549–560. ACM (2013)
15. Lapets, A., Varia, M., Bestavros, A., Jansen, F.: Role-based ecosystem model for design, development, and deployment of secure multi-party data analytics applications. Cryptology ePrint Archive (2017)
16. Levy, A.: Fraudsters just stole $7M by hacking a cryptocoin offering. https://www.cnbc.com/2017/07/17/coindash-website-hacked-7-million-stolen-in-ico.html, [Accessed: August 24, 2017]
17. Morton, B.: Code Signing. https://casecurity.org/wp-content/uploads/2013/10/CASC-Code-Signing.pdf, [Accessed: July 13, 2017]
18. Ptacek, T.: Javascript Cryptography Considered Harmful. https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/, [Accessed: July 11, 2017]
19. Shamir, A.: How to share a secret. Comm. of the ACM 22(11), 612–613 (1979)
20. Yao, A.C.: Protocols for secure computations. In: Proc. of the 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164. IEEE Comp. Soc. (1982)