

DEMO: Secure Computation in JavaScript

Axel Schröpfer
SAP Research
Karlsruhe, Germany
axel.schroepfer@sap.com

Florian Kerschbaum
SAP Research
Karlsruhe, Germany
florian.kerschbaum@sap.com

ABSTRACT

Secure computation, e.g. using Yao’s garbled circuit protocol, allows two parties to compute arbitrary functions without disclosing their inputs. A profitable application of secure computation is business optimization. It is characterized by a monetary benefit for all participants and a high confidentiality of their respective input data. In most instances the consequences of input disclosure, e.g. loss of bargaining power, outweigh the benefits of collaboration. Therefore these optimizations are currently not performed in industrial practice.

Our demo shows such an optimization as a secure computation. The joint economic lot size (JELS) is the optimal order quantity between a buyer and supplier. We implemented Yao’s protocol in JavaScript, such that it can be executed using two web browsers. This has the additional benefit that the software can be offered as a service (SaaS) and can be easily integrated with other SaaS offerings, e.g. using mash-up technology.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Cryptographic controls*; C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*

General Terms

Security

Keywords

Secure Computation, JavaScript, Mash-up, Business Optimization

1. INTRODUCTION

Secure computation [8] allows two parties to compute an arbitrary function on joint inputs without disclosing anything except what can be inferred by one party’s input and output. This offers an intriguing solution to many real-world problems where collaboration is prevented by the reluctance to disclose one’s data. In business optimization there are many computations with a mutual, monetary benefit for the collaborators. Examples from the supply chain management literature include [1, 7, 5].

However, they all require the input of very sensitive data, e.g., production costs or capacities. The threat of an abuse of this data outside the intended purpose, e.g. in a future negotiation resulting in a loss of bargaining power, prevents the implementation of these computations even using a trusted third party [7]. Secure computation can help to overcome this dilemma. We describe our use case in Section 3.

Current implementations of secure computation presume that all participants operate their own equipment. This is not true for modern business software. Most companies offer their software as a service, e.g. salesforce.com or SAP Business ByDesign. We therefore implemented the secure computation entirely in JavaScript.

In this demo we

- (i) introduce a representative, meaningful use-case from industrial practice in supply chain management consisting of an arithmetic function on a buyer’s and supplier’s sensitive inputs.
- (ii) present an architecture for secure computations in JavaScript over the web
- (iii) show an implementation which allows to execute many arithmetic formulas from business optimization with very little additional effort.

2. WEB APPLICATION SYSTEM

Yao’s garbled circuit protocol [8] allows arbitrary secure computations for two parties. This protocol can be implemented in two rounds and is secure in the semi-honest model. In the semi-honest model the participants follow the protocol, but try to infer additional information from the messages received. The semi-honest model is well suited for business optimization where the participants have an interest in receiving the correct result.

Several secure computation frameworks and programming languages have already implemented Yao’s protocol. Fairplay [4] was the first practical secure computation, but soon others followed [2, 6]. All these implementations presume the software to be executed on local premises. To the contrary – motivated by current software-as-a-service offerings and future scenarios such as mobile devices – we implement a secure computation application running in a common web browser using only JavaScript.

Our system consists of two web browsers, one at each party’s side. Each browser runs a Yao’s protocol run-time environment implemented entirely in JavaScript. This run-time executes a circuit which is represented in JavaScript



Figure 1: Web application system for Yao's protocol.

Object Notation (JSON). We furthermore provide a generator which creates this JSON circuit object for many, to be specified arithmetic formulas. It translates the formula consisting of basic arithmetic operations (addition, subtraction, multiplication and division) into a combinational circuit by composing (i.e., wiring) corresponding sub-circuits.

OT protocols – including the variant used by us – require public key cryptography and therefore multi-precision integers. Furthermore, the operation of garbling (and later evaluating) the circuit in Yao's protocol require extensive calls to a hash function (e.g., SHA-1). Neither are readily available in JavaScript. We use the implementation of *BigInt* by Leemon Baird¹ as a replacement for Java's *BigInteger* class and our own SHA-1 implementation based on *BigInt* (which turned out to perform best among a group of evaluated alternatives).

Unfortunately, we were not able to leverage parallel computation. Although the JavaScript API supports threading, CPUs utilization revealed that the underlying scheduling is still insufficient. Another obstacle when using JavaScript is generating truly random numbers with the built-in generator. This issue can be best resolved using browser-specific functions.

The communication between the two run-time environments in the web browsers is realized by the central HTTP server relaying messages. Fig. 1 shows screenshots of our system.

Our system is capable of securely implementing a wide range of business optimizations. Many such optimizations can be represented as simple arithmetic formulas for two parties' inputs. We thereby enable the practical adoption of these optimizations in modern (SaaS) business software. The JavaScript implementation allows the easy consumption and integration of our services, e.g. using mash-up technology.

3. USE-CASE: JOINT ECONOMIC LOT SIZE

We consider a simple supply chain scenario consisting of a single supplier (Alice) and a single buyer (Bob) of a specific product. The buyer and the supplier have negotiated a fixed

¹<http://www.leemon.com/crypto/BigInt.html>

supply quantity of d units per period, e.g. a year. Let q_B denote the order quantity determined by the buyer. It is assumed that the supplier has sufficient capacity to fulfill the buyer's orders and that his lead time is zero. The buyer's total relevant costs $TRC_B(q_B)$ are

$$TRC_B(q_B) = \frac{d}{q_B} \cdot f_B + \frac{1}{2} \cdot q_B \cdot h_B,$$

where f_B denotes the buyer's fixed cost per order, and h_B his inventory holding cost per unit and period. $\frac{1}{2} \cdot q_B$ is the average inventory throughout the period. The buyer's economic order quantity q_B^* is

$$q_B^* = \sqrt{2 \cdot d \cdot \frac{f_B}{h_B}}.$$

Let q_A denote the supplier's production lot size, c the production capacity per period (with $c \leq d$), f_A the set-up cost per production lot, and h_A his inventory holding cost per unit and period. Assume that the supplier follows a lot-for-lot production policy; he cannot accommodate lot streaming and delivers the entire production batch after its completion. Thus, inventory only occurs throughout each production cycle and drops to zero after a lot is completed. The supplier's total relevant costs, denoted by $TRC_A(q_A)$ are

$$TRC_A(q_A) = \frac{d}{q_A} \cdot f_A + \frac{1}{2} \cdot q_A \cdot \frac{d}{c} \cdot h_A$$

and his economic lot size is

$$q_A^* = \sqrt{2 \cdot c \cdot \frac{f_A}{h_A}}.$$

Under the assumption of a lot-for-lot production policy, the lot size of the supplier will correspond to the order size of the buyer, i.e., $q_A = q_B$. In most cases $q_A = q_B^*$, resulting in total joint costs of

$$JTRC(q_B^*) = TRC_A(q_B^*) + TRC_B(q_B^*).$$

Based on this brief outline of the problem setting, it is easy to explain the underlying rationale of the JELS model: if

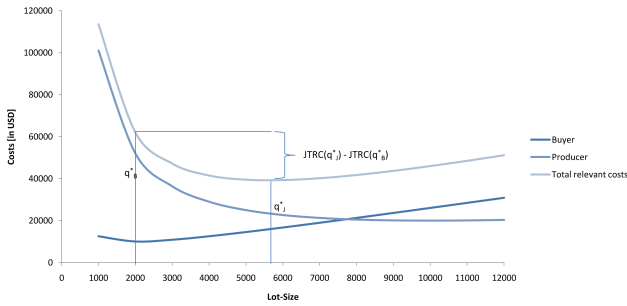


Figure 2: JELS Sample - Overall costs reduce from \$62.000 to \$39.250.

```

1  default -bits :32
2
3  _2dfA :Bob
4  hA :Bob
5  _2dfB :Alice
6  dhB_c :Alice
7  qJstar2 :Bob, Alice
8
9  qJstar2 = (_2dfB + _2dfA) / (dhB_c + hA)

```

Listing 1: JELS formula for JSON generator.

$q_B^* \neq q_A^*$ there exists a “joint economic lot size”, denoted q_J^* , that leads to minimal joint costs of the buyer and supplier such that $JTRC(q_J^*) < JTRC(q_B^*)$. The joint economic lot size q_J^* [1] is given by

$$q_J^* = \sqrt{2 \cdot d \cdot \frac{f_A + f_B}{d \cdot \frac{h_A}{c} + h_B}}. \quad (1)$$

Fig. 2 depicts an example for this scenario. For the supplier we have $f_A = 1000$ \$/unit, $h_A = 3$ \$/unit and $c = 150.000$ units. For the buyer we have $f_B = 100$ \$/unit, $h_B = 8$ \$/unit. They have agreed upon $d = 100.000$ units per period. The buyer’s optimal quantity is $q_B^* = 2.000$ leading to total costs of $JTRC(q_B^*) = \$62.000$. Using the joint economic lot size $q_J^* = 5500$ leads to total costs of $JTRC(q_J^*) = \$39.250$.

4. IMPLEMENTING JELS IN THE BROWSER

Using our web application system for Yao’s protocol (Section 2) we derive a web application for the use case of collaboratively computing JELS (Section 3). First, in order to save (significant) computation time we optimize the original textbook formula for q_J^* Eq. (1) to

$$q_J^{*2} = \frac{a + b}{c + d}$$

by setting $a = 2 \cdot d \cdot f_B$, $b = 2 \cdot d \cdot f_A$, $c = d \cdot h_B$ and $d = h_A$ as the local inputs [3]. This way some computations can be performed offline before and after the secure computation. This optimization reduces the size of the circuit and consequently the run-time.

We can then specify the formula in the format required for the JSON object circuit generator, as depicted in Lst. 1.

We first define the bit-length of the variables (line 1) and then we define variables and their respective owners in a *variablename : ownername* notation (lines 3-7). Finally, we specify the formula (line 9).

We generate the JSON object and copy it to the directory hosting the HTML pages for gathering inputs, wrapping the JavaScript code and displaying outputs – i.e. the run-time environment. Usually, this is also the web server hosting the message relay service.

Executing the secure JELS computation (without manual optimizations of the circuit) on a common office PC with Mozilla Firefox we obtain a run-time of roughly 300 seconds.

5. CONCLUSIONS

We demonstrate how secure computations can be performed as software-as-a-service (SaaS) using a standard web browser. Our system allows to securely compute a large class of two-party business optimization functions in JavaScript. Hence, we enable the realization of many, new saving potentials in modern SaaS business software applications.

We also demonstrate the representative and meaningful use case of the joint economic lot size. Using our generator it is easy to implement the necessary functionality. Our application is consumable using only standard software, available on a large range of platforms including mobile devices.

6. REFERENCES

- [1] A. Banerjee. A joint economic-lot-size model for purchaser and vendor. *Decision Sciences*, 17(3):292–311, 1986.
- [2] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS ’10, 2010.
- [3] F. Kerschbaum. Automatically optimizing secure computation. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.
- [4] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - a secure two-party computation system. In *Proceedings of the USENIX security symposium*, 2004.
- [5] R. Pibernik, Y. Zhang, F. Kerschbaum, and A. Schröpfer. Secure collaborative supply chain planning and inverse optimization - the jels model. *European Journal of Operational Research*, *EJOR*, 208(1):75–85, 2011.
- [6] A. Schröpfer, F. Kerschbaum, and G. Müller. L1 - an intermediate language for mixed-protocol secure computation. In *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference*, *COMPSAC*, 2011.
- [7] H. Stadtler. A framework for collaborative planning and state-of-the-art. *OR Spectrum*, 31:5–30, 2009.
- [8] A. Yao. How to generate and exchange secrets. In *In Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.