On the Practical Importance of Communication Complexity for Secure Multi-Party Computation Protocols

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

Daniel Dahlmeier
SAP Research
Karlsruhe, Germany
daniel.dahlmeier@sap.com

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

Debmalya Biswas*
IRISA/INRIA
Rennes, France
dbiswas@irisa.fr

ABSTRACT

Many advancements in the area of Secure Multi-Party Computation (SMC) protocols use improvements in communication complexity as a justification. We conducted an experimental study of a specific protocol for a real-world sized problem under realistic conditions and it suggests that the practical performance of the protocol is almost independent of the network performance. We argue that our result can be generalized to a whole class of SMC protocols.

Categories and Subject Descriptors

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

General Terms

Security, Experimentation, Performance, Measurement

Keywords

Secure Multi-Party Computation, Experimentation, Performance Measurement, Synchronization

1. INTRODUCTION

Secure Multi-Party Computation (SMC) protocols offer a general solution for many problems in secure distributed computing, e.g. private auctions or joint database calculations. The practical realization of SMC protocols is currently emerging, as problems of practical size and interest come into reach [5, 9, 17]. On the other hand the theoretical development and research is continuing. Many of the theoretical results use improvements in communication complexity as justification and a measure of success. This seems logical as the communication complexity is the lower bound of the computation complexity.

We have conducted an experimental study of a SMC protocol for privacy-preserving benchmarking [1, 15]. Privacy-preserving benchmarking is an important collaborative business application and of great economic interest. The goal of the study was to evaluate the practical feasibility of real-world sized privacy-preserving benchmarking under realistic conditions. Its conclusion besides the practical feasibility is that the absolute performance is almost independent of the network performance, i.e. the protocol runs as fast if the participant are joint by a local area network (LAN) as if the participant are distributed over a wide area network (WAN).

Our privacy-preserving benchmarking protocol has special properties differentiating it from general purpose SMC protocols. Our protocol has a central communication pattern, i.e. all clients communicate only with a central server that performs most of the computation. Furthermore, our protocol is organized in rounds: Each participant must complete the steps within one round before any participant can begin the next round. This implies a certain distributed synchronization pattern of our protocol. We call this pattern "centrally coordinated", since a central server coordinates the protocol.

We show with the example of the landmark protocol by Ben-Or, Goldwasser and Wigderson [3] (BGW) that general SMC protocols can be implemented centrally coordinated. This is to stress that SMC protocol performance is computation bound and not network bound. We believe that this justifies a closer look at the performance characteristics of theoretic results.

The remainder of the paper is structured as follows: We review the privacy-preserving benchmarking protocol in the next Section. Then we present the implementation and results of the experimental study. The next Section analyses the distributed synchronization pattern and shows the modifications to the BGW protocol for central coordination. We review some related work before the last Section presents the conclusions of the paper.

*The work was performed while the author was at SAP Research.

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

2. PRIVACY-PRESERVING BENCHMARKING

Benchmarking is the comparison of one company's key performance indicators (KPI) to the statistics of the same KPIs of its peer group. A key performance indicator (KPI) is a statistical quantity measuring the performance of a business process. Examples from different company operations are make cycle time (manufacturing), cash flow (financial) and employee fluctuation rate (human resources). A peer group is a group of (usually competing) companies that are interested in comparing their KPIs based on some similarity of the companies. Examples formed along different characteristics include car manufacturers (industry sector), Fortune 500 companies in the United States (revenue and location), or airline vs. railway vs. haulage (sales market).

Privacy is of the utmost importance in benchmarking. Companies are reluctant to share their business performance data due to the risk of losing a competitive advantage or being embarrassed. Several privacy-preserving protocols that can be used for benchmarking that keep the KPIs confidential within one company have been proposed in the literature [1, 5, 9, 15]. None of those matches the requirements of a large service provider offering a benchmarking service entirely. Instead we designed a new privacy-preserving benchmarking protocol.

This protocol must fulfill the following requirements. Its clients must remain *anonymous* amongst each other. This can be achieved by *central communication*, i.e. all clients only communicate with one central server, and cCommunication must be *free of identifiers* of clients. This includes pseudonyms, such as static (public) keys. Furthermore it must compute these *stochastic metrics*: average, variance, maximum, median, best-in-class (average of top quarter). The protocol must also be *private* in the semi-honest setting [10].

2.1 Preliminaries

2.1.1 Homomorphic Encryption

Our protocol is built using homomorphic encryption. In homomorphic encryption one operation on the ciphertexts produces an encryption of the result of a homomorphic operation on the plaintexts. In particular, we require the homomorphic operation to be addition (modulo a key-dependent constant). Several such encryption systems exist [2, 7, 18, 21, 22]. We used Paillier's encryption system [22] in the implementation. Let $E_X(x)$ denote the encryption of x with X's public key and $D_X()$ the corresponding decryption with X's private key, then Paillier's encryption system has the following property:

$$D_X(E_X(x) \cdot E_X(y)) = x + y$$

With simple arithmetic the following property can be derived

$$D_X(E_X(x)^y) = x \cdot y$$

2.1.2 Key Distribution

Our implementation uses a simple protocol to perform the following key distribution: Each client X_i has access to a common private key in the homomorphic encryption scheme. The server does not have access to the private key; he can only access the public key. As a consequence, each client X_i can perform the operations $D_{common}()$ (decryption) and $E_{common}()$ (encryption), while the server can only perform the operation $E_{common}()$. Decryption is the (left-)inverse operation to encryption, i.e. $D_{common}(E_{common}(x)) = x$.

Each client X_i has access to a further secret key s_{common}. This is a key for a message authentication code and is also kept secret from the server.

The details of the key distribution protocol are omitted, but it is executed during the registration of clients. It is only run once for each client using a modified PKI certificate authority and does not influence the performance of the main statistics computation protocol.

2.1.3 Oblivious Transfer

As another building block our protocol uses Oblivious Transfer (OT). OT was introduced in [23] and generalized to 1-out-of-2 OT in [8]. In 1-out-of-2 OT the sender has two secrets x_0 and x_1 and the receiver has one bit b. After the execution of the OT protocol, the receiver has obtained x_b, but has learnt nothing about $x_{\neg b}$, and the sender has not learnt b. The fastest known implementation of OT which is also used in our implementation is described in [19]. The efficient version of the protocol was proven secure under the (computational and decisional) Diffie-Hellman and the random oracle model assumptions. The authors also give a less efficient construction secure under the decisional Diffie-Hellman assumption alone which we did not implement. An OT protocol between a sender S and a receiver R (where the parameters are clear from the context) is denoted by

$$S \xrightarrow{\text{OT}} R$$

2.2 Protocol

The protocol is a composition of several techniques. Let x_i be the KPI, i.e. the input, of participant X_i. In the first round each participant X_i submits his input x_i encrypted under the common homomorphic key. The server then chooses two random numbers r and r' for each pair x_i, x_j of inputs, such that $r > 0$ and $r > r' \geq 0$ and computes a value c_{i_j} as

$$E_{common}(c_{i_j}) =$$

$$(E_{common}(x_i) \cdot E_{common}(x_j)^{-1})^r \cdot E_{common}(r') =$$

$$E_{common}(r \cdot (x_i - x_j) + r')$$

As long as wrap-around is prevented, it holds that

$$c_{i_j} < 0 \Leftrightarrow x_i < x_j$$

The value c_{i_j} does not reveal the hidden values x_i, x_j or their difference. It is hidden by the multiplicative factor r and in order to prevent factoring r' has been added. In the proof of security we assume that c_{i_j} completely hides $x_i - x_j$. We chose to accept this assumption, because this method of comparison is particularly efficient. Since this only affects the computational efficiency and not the communication complexity, it does not affect the main thesis of this paper.

In the second round, each participant X_i is given a random chosen vector $\vec{c}_{\Phi(i)} = (c_{\Phi(i)_1}, \ldots, c_{\Phi(i)_n})$ according to permutation $\Phi(i)$ chosen by the server. Actually the server sends the encryption of the elements of vector $\vec{c}_{\Phi(i)}$, but X_i can decrypt them to obtain $\vec{c}_{\Phi(i)}$. The number of non-negative elements in $\vec{c}_{\Phi(i)}$ indicates the rank of element $x_{\Phi(i)}$.

Round 1:

$X_i \longrightarrow SP \quad E_{common}(x_i)$

Round 2:

$SP \longrightarrow X_i \quad E_{common}(sum + r_1) = E_{common}(\sum_{i=1}^{n} x_i) \cdot E_{common}(r_1)$

$E_{common}(c_{\vec{\Phi(i)}}) = (\ldots, E_{common}(c_{\Phi(i)_l}) = E_{common}(r_{2_l} \cdot (x_{\Phi(i)} - x_{\Phi(l)}) + r_{3_l}), \ldots)$

$SP \xrightarrow{\text{OT}} X_i \quad E_i^{median} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{4_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = \lceil \frac{n}{2} \rceil \\ E_{common}(r_{4_i}) & \text{otherwise} \end{cases}$

$E_i^{quart} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{5_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = \lceil \frac{3}{4} n \rceil \\ E_{common}(r_{5_i}) & \text{otherwise} \end{cases}$

$E_i^{max} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{6_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = n \\ E_{common}(r_{6_i}) & \text{otherwise} \end{cases}$

$X_i \longrightarrow SP \quad sum + r_1$

$MAC(sum + r_1 | i, s_{common})$

$E_i^{median} \cdot E_{common}(0)$

$E_i^{quart} \cdot E_{common}(0)$

$E_i^{max} \cdot E_{common}(0)$

$E_{common}((x_i - \frac{sum}{n})^2)$

$SP \longrightarrow X_i \quad sum$

Round 3:

$SP \quad E_{common}(quart) = \prod_{i=1}^{n} E_i^{quart} \cdot E_{common}(-r_{5_i})$

$SP \longrightarrow X_i \quad E_{common}(sum' + r_7) = E_{common}(\sum_{i=1}^{n} (x_i - avg)^2) \cdot E_{common}(r_7)$

$E_{common}(median + r_8) = (\prod_{i=1}^{n} E_i^{median} \cdot E_{common}(-r_{4_i})) \cdot E_{common}(r_8)$

$E_{common}(max + r_9) = (\prod_{i=1}^{n} E_i^{max} \cdot E_{common}(-r_{6_i})) \cdot E_{common}(r_9)$

$H(MAC(sum + r_1 | 1, s_{common}), \ldots, MAC(sum + r_1 | n, s_{common}))$

$E_{common}(c) = E_{common}(r_{10} \cdot (x_{\Phi(i)} - quart) + r_{11})$

$SP \xrightarrow{\text{OT}} X_i \quad E_i^{bic} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{12_i}) & \text{if } c \geq 0 \\ E_{common}(r_{12_i}) & \text{if } c < 0 \end{cases}$

$X_i \longrightarrow SP \quad sum' + r_7$

$MAC(sum' + r_7 | i, s_{common})$

$median + r_8$

$MAC(median + r_8 | i, s_{common})$

$max + r_9$

$MAC(max + r_9 | i, s_{common})$

$E_i^{bic} \cdot E_{common}(0)$

$SP \longrightarrow X_i \quad sum'$

$median$

max

Round 4:

$SP \longrightarrow X_i \quad E_{common}(bic + r_{13}) = (\prod_{i=1}^{n} E_i^{bic} \cdot E_{common}(-r_{12_i})) \cdot E_{common}(r_{13})$

$H(MAC(sum' + r_7 | 1, s_{common}), \ldots, MAC(sum' + r_7 | n, s_{common}))$

$H(MAC(median + r_8 | 1, s_{common}), \ldots, MAC(median + r_8 | n, s_{common}))$

$H(MAC(max + r_9 | 1, s_{common}), \ldots, MAC(max + r_9 | n, s_{common}))$

$X_i \longrightarrow SP \quad bic + r_{13}$

$MAC(bic + r_{13} | i, s_{common})$

$SP \longrightarrow X_i \quad bic$

Round 5:

$SP \longrightarrow X_i \quad H(MAC(bic + r_{13} | 1, s_{common}), \ldots, MAC(bic + r_{13} | n, s_{common}))$

Figure 1: Benchmarking protocol

Several of the sought-after quantities can be defined via the rank: The maximum has rank n, the median has rank $\lceil \frac{n}{2} \rceil$, and the least element still included in the best-in-class computation has rank $\lceil \frac{3n+3}{4} \rceil$. The server and the participants must now compute the value of these elements. Participant X_i has the rank of KPI $x_{\Phi(i)}$, but does not know the value $x_{\Phi(i)}$. The server has $E_{common}(x_{\Phi(i)})$, chooses a random number r and prepares two values for Oblivious Transfer: $E_{common}(x_{\Phi(i)} + r)$ and $E_{common}(r)$. In the OT the receiver X_i chooses $E_{common}(x_{\Phi(i)} + r)$ if $x_{\Phi(i)}$ has the selected rank, otherwise he chooses $E_{common}(r)$. Note that due to the secret sharing using the random number r, X_i has learnt nothing about $x_{\Phi(i)}$ and he can return the chosen value immediately to the server. The server adds r using the homomorphic operation and after the round sums up all received values also using the homomorphic operation. He ends up with a ciphertext of the element with a specific rank. This is done for all three quantities: maximum, median and the least best-in-class element.

In the third round, the server repeats the comparison operation, but only against the least best-in-class element. In the same procedure using OT all elements equal or larger to this element are chosen and summed up.

The server ends up with ciphertext of the results for each quantity, but he does not have the decryption key. If he submits, the ciphertext to only one participant, this participant can prevent all other participants from obtaining the correct result while still obtaining it himself by returning an incorrect value. A Zero-Knowledge-Proof of correct decryption would not prevent the server from submitting to all clients.

If he submits the result to all clients, the server can cheat and submit the original input to each client and compute the result from the plaintext. He would then have broken the security of the protocol without modifying the result of the computation. The server therefore sends a proof to all clients that he submitted the same value for decryption to all clients. The clients X_i sign the value with a (personalized) message authentication code and the server computes an aggregate signature that all clients can verify. This prevents the server from deviating from the protocol without modifying the result. A complete proof of this security property can be found in [14]. The result is hidden before decryption using secret sharing, such that the server can round the result to an appropriate level for disclosure.

Figure 1 gives a formal description of the entire protocol. It describes the two-party interaction between a client X_i and the server SP subdivided into rounds. Each round needs to be completed by each client X_i $(i = 1, \ldots, n)$ before any client can engage in the next round. The order of the clients is not important.

2.2.1 Analysis

The protocol has communication complexity $O(n^2)$: Each of the n clients communicates $O(n)$ data items. This happens in round 2 when client X_i receives the vector $\vec{c}_{\Phi(i)}$, as all other communication is constant ($O(1)$). Computation complexity is $O(n^2)$ for the server and $O(n)$ for the clients. The server has to prepare all n vectors $\vec{c}_{\Phi(i)}$ of size n for round 2 doing a constant number of modular exponentiations for each, while all other preparation steps are linear. The client has to decrypt all n entries in his vector $\vec{c}_{\Phi(i)}$ doing a constant number of modular exponentiations for each. Let k be a security parameter, then the computation complexity is $O(n^2k^3)$ for the server and $O(nk^3)$ for the clients. This is due to modular exponentiation for homomorphic encryption and decryption.

The protocol is secure in the semi-honest setting following the definitions of [10], i.e. loosely speaking the participants follow the protocol, but keep a record in order to compute additional information. The proof is standard by giving a simulator of the message received by either client or server. It is important to note that the server is treated just as another party, i.e. the proof is for an $n + 1$ party protocol. One assumption is that a number hidden by multiplying and adding a random number is effectively hidden. This manifests itself in the assumption that this number can be simulated using a random number chosen from a distribution independent of the number hidden.

3. IMPLEMENTATION

We implemented the protocol based on web services as a communication mechanism. Each client and server runs its own web application server with a web application for the protocol implementation. The implementation is entirely in Java with the crypto modules taking advantage of the BigInteger arithmetic of Java's standard library.

The communication between clients X_i and server SP follows a client-server model with each client X_i in the protocol being a client at the communication level and the server SP being the server. Each communication is sent via HTTP as the binding protocol for SOAP (and web services) over a TCP connection, i.e. the client X_i is the requestor in HTTP and the client in TCP and the server SP is server in HTTP and TCP. The client then makes several invocations of web services at the server's side which transport as parameters and response types the data items described in Figure 1.

Each round of the protocol is implemented as its own class later representing a web service client and web service (application). The server hosts the web service for each of the rounds.

The server orchestrates the protocol, i.e. it signals each client X_i when he has to start a round of the protocol. The server first creates a web service session with its web service application. This server session is given access to a special server object that hosts the specific instance variables for this client. An example of such instance variables are the permuted vectors $\vec{c}_{\Phi(i)}$ at the beginning of round 2. Each round 2 server object is given one such vector $\vec{c}_{\Phi(i)}$ and associated with a specific session. The server then calls a web service at a client X_i that triggers the client's execution of the web service client, i.e. the server acts as a client on HTTP and TCP level here calling a single-instance, stateless web service at the client X_i (which acts as a server on HTTP and TCP level). As a parameter in this call the server passes the session id to the client. The client then creates a web service stub for precisely this session and thereby addresses the corresponding instance of server object on the server's side. All communication is encrypted and authenticated using WS-Security such that sensitive information such as session ids is not being leaked, i.e. there are secure and authenticated channels between the server and clients.

The overall communication looks as in Figure 3 where an example of the communication for round 2 is given without the OT which are implemented as sub-protocols similar to rounds. Requests are drawn as full lines and responses as dotted lines. Each thread is given its own box and swim lanes are reserved for parties in the distributed protocol.

The server orchestrates the entire benchmarking protocol by executing one round with each client in turn. The server has a loop in which he creates the server session, calls the client and destroys the server session, for each client and each round. If the client loop is the inner loop (and the round loop the outer loop) the coordination requirement that each client must complete round j before round $j + 1$ can be started is ensured. Between two rounds the server prepares the input for the next round. E.g. he computes the vectors $\vec{c}_{\Phi(i)}$ between round 1 and 2. The communication for two rounds and three clients looks as in Figure 2. The client and server execute a communication pattern as in Figure 3 for each of the clients' boxes.

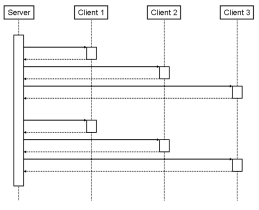

Figure 2: Example communication of three clients with the server in two rounds

We evaluated this implementation in an experimental study. The server was deployed on a Pentium 4 3.2 GHz machine with 1.5 GB of memory. All clients were deployed on a Xeon

Dual 3.6 GHz machine with 8 GB of memory. Between the client and server machine we deployed a WAN emulator as an IP router. The WAN emulation software was the dummynet package for FreeBSD [24]. All machines are physically connected via a non-dedicated Gigabit Ethernet switch.

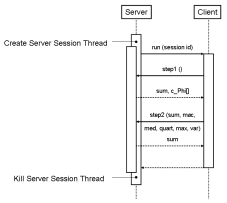

Figure 3: Example communication of one client and server within round 2 without OT

We independently modified two parameters: We increased the number of clients n from 5 to 45 clients in steps of 5 and we increased the latency on the network connection from 0 to 100 milliseconds in steps of 25. The latency or delay is used to simulate WAN conditions as over the Internet. A delay of 100 ms results in a round-trip time (RTT) of 200 ms, which roughly corresponds to the RTT between Germany and Japan over the Internet. RTTs to destinations in the US are shorter from Germany and RTTs to destinations in Europe are even shorter than that.

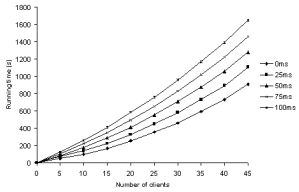

Figure 4: Running time of the protocol in dependence on number of clients and network delay

The results are depicted in Figure 4. It is apparent from this picture that in this implementation the network performance plays a significant role. For 45 clients and a delay of 100 ms the time spent for communication is almost half (precisely 45%) of the overall running time. The average for clients from 5 to 45 is 54% and constantly decreasing. This can be expected, since the computational complexity is $O(n^2)$ while the number of connections to clients (incurring the delay) is $O(n)$. Therefore in the limit the computational performance will be dominating. Nevertheless for these real-world number of clients the time spent on the network is significant.

For the next experiment we modified the server implementation. Instead of sequentially calling each client X_i in a loop, we create a thread for each client that asynchronously handles the communication. This is possible in this version of the benchmarking protocol, since each round for each client only requires input of the previous round. i.e. all clients can run concurrently. As we pointed out before the order of the clients does not matter for the protocol's semantics. The necessary synchronization can be achieved using a barrier. The barrier synchronizes $n+1$ threads: each thread communicating with a client and the main thread. A thread calls the barrier object method and waits in sleeping mode until it is tripped. The last thread reaching the barrier, trips it and all threads continue. The main thread continues to the next round after waiting for the barrier while the other threads exit. The client threads have finished communication before they reach the barrier. We used the barrier implementation from Java's standard library module for concurrent utilities. The resulting communication is shown in Figure 5. Each client and client thread at the server's side execute a communication pattern as in Figure 3 again.

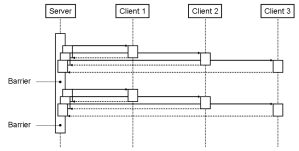

Figure 5: Example concurrent communication of three clients with the server in two rounds

We conducted the same series of experiments for the concurrent implementation. We increased the number of clients and independently increased the network delay.

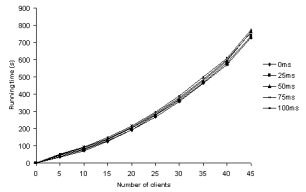

Figure 6: Running time of the protocol in dependence on number of clients and network delay in concurrent implementation

The results are depicted in Figure 6. The impact of the network performance has significantly decreased and is almost negligible compared to the impact of the computational effort. For 45 clients a delay of 100 ms the time spent for communication is only 6% of the overall running time. The average for clients from 5 to 45 is 14%. We therefore conclude that the benchmarking protocol can be implemented, such that its performance is almost independent of the network performance, i.e. for the overall performance it nearly does not matter whether the clients are located on the same LAN or half-way around the world over the Internet.

The difference of the running time spent on communication can be explained with the different synchronization patterns of the sequential and concurrent implementation. In the sequential implementation for each client a time period t_c is being spent for communicating the request from the server to the client and the response from the client to the server. This time t_c is dominated by the latency of the network connection. During this time neither client nor server can perform other computations or communications and a client has to wait until all its predecessor have finished. The running time spent on the network is therefore dominated by a linear number of delays due to the latency of the

network. In the experiment using the sequential implementation the time spent on communication increased linearly with the number of clients supporting this hypothesis.

In the concurrent implementation the time spent on one round is dominated by the slowest client. Since, in our implementation all clients are identical (with identical network characteristics) the time is dominated by one client. If the server is not able to schedule clients sufficiently fast, communications overlap only partially, but the communication with one client may happen while another client is computing. Therefore the communication time is dominated by the delay as incurred in every round of the protocol.

A limitation of our study is that computation time far exceeds the delay on the network. In our benchmarking protocol each client and the server computes on the order of minutes while the delay of the network is on the order of tens of milliseconds. This may shadow the impact of network performance compared to protocols where less computation is performed.

We also compared the secure distributed computation to a local computation of the same functionality. For 45 clients the benchmarking functionality can locally be implemented in Java in 15 ms as opposed to 15 minutes for the secure distributed computation. This corresponds to a slow-down of roughly 60.000 for the secure distributed computation. Obviously a trusted third party would incur a penalty for secure data transportation, storage and retrieval, but we believe this factor is still the best indicator for a practical secure SMC protocol.

4. GENERALIZATION

So far, we have shown that the benchmarking protocol's performance is almost independent of the network performance, but does this apply to other SMC protocols as well? The claim is that if they follow the same communication and synchronization pattern, they exhibit similar performance characteristics.

4.1 Central coordination

The benchmarking protocol follows a communication and synchronization pattern we call "centrally coordinated". All communication is between a client X_i and the server SP. The server triggers all computation and communication on the clients, i.e. the clients can be implemented single threaded. There is a requirement by the semantics of the protocol that a particular computation step needs to be completed by all clients and the results communicated before any client can begin the next computation step. Such a computation step is called a round, since similar to the synchronous network model of distributed systems [16] these steps are performed in lock-step by the clients and the server. Within each round a client may communicate multiple times with the server. The server must synchronize all clients, i.e. wait for their completion, before he can initiate the next round. This is the only synchronization requirement. We formally define:

DEFINITION 1. *Let Π be a n-party protocol with clients X_i ($i = 1, \ldots, n$). Π is* centrally coordinated *if there is a $n+1$-th party SP called server, such that*

- *each client X_i only communicates with the server SP*
- *the server SP triggers all computation and communication of all clients X_i*
- *the server SP solely synchronizes the communication with the clients X_i*
- *there is no dependency among clients between two synchronization steps.*

4.2 Other SMC Protocols

We describe how to implement the BGW protocol [3] using a central coordinator. The BGW was the first SMC protocol secure in the information-theoretic setting, i.e. it does not use encryption. Instead it used secret sharing, e.g. using Shamir's secret sharing scheme [25]. In this secret sharing scheme a secret s is distributed among n parties using a polynomial of degree $t-1$:

$$f(x) = s + a_1 x + \ldots + a_{t-1} x^{t-1}$$

Each party X_i is assigned an identifying index α_i and given a share $f(\alpha_i)$. Then t parties can reconstruct the secret s by LaGrange interpolation. We call such a secret sharing scheme a (t, n) threshold secret sharing scheme.

In the BGW protocol each (input, output or intermediate) value is shared using a (t, n) Shamir's secret sharing scheme ($t < \frac{n}{3}$). Let $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ be the shares for value a and b distributed over the n parties X_i, i.e. X_i has x_i and y_i. The parties can compute the sum of a and b as

$$a + b : \vec{x} + \vec{y}$$

No communication is necessary between the parties. They can also compute the product of a with a constant c without communication:

$$ca : c\vec{x}$$

Computing the product of a and b is more complicated. First, each party X_i computes $g_i = x_i y_i$, which are shares of a $2t-2$ degree polynomial $g(x)$.

$$ab : \vec{g} = (x_1 y_1, \ldots, x_n y_n)$$

Second, each party X_i chooses a random $2t-2$ degree polynomial $h_i(x)$ with constant term 0 and sends $h_{i,j} = h_i(\alpha_j)$ to party X_j. The parties compute the sum of the random polynomials and $g(x)$.

$$\vec{h} = (g_1 + \sum_{i=1}^{n} h_{i,1}, \ldots, g_n + \sum_{i=1}^{n} h_{i,n})$$

The main theorem of [3] shows that the shares $\vec{h}$ of a random $2t-2$ degree polynomial can be reduced to shares $\vec{k}$ of a random $t-1$ degree polynomial by multiplication with a constant $n \times n$ matrix M known to all parties.

$$\vec{k} = \vec{h}M$$

Each participant X_i computes locally

$$\vec{k_i} = (k_{i,1}, \ldots, k_{i,n}) = h_i(m_{i,1}, \ldots, m_{i,n})$$

Participant X_i sends $k_{i,j}$ to participant X_j and they jointly compute

$$\vec{k} = (\sum_{i=1}^{n} k_{i,1}, \ldots, \sum_{i=1}^{n} k_{i,n})$$

The participants continue computation using the shared value ab with shares $\vec{k}$ on a random $t-1$ degree polynomial.

The BGW protocol requires secure channels between all parties, i.e. if we simulate these channels using encryption, security is reduced to the computational setting. In the centrally coordinated model all communication between X_i and X_j ($i \neq j$) is channeled through the server, i.e. the server has total control of the network. In order to keep the server from learning the inputs in the semi-honest setting, we can use encryption to establish secure channels between X_i and X_j. For security against malicious servers, we need to also add message authentication, round numbering and a secure session id.

Let $E_i()$ denote the encryption with X_i's public key. The public key of each client X_i has been securely distributed to all other clients, e.g. using a public-key infrastructure. Figure 7 shows the BGW protocol step for multiplication with central coordination.

Since addition does not require communication the parties can optimize the overall protocol in several ways. First they combine round 3 and round 1 of two consecutive multiplication protocols. Second they run several multiplication protocols in parallel for gates that do not depend on each other, i.e. whose input and output are not connected by (directed) wires. This reduces the round complexity to $2d + 3$ where d is the multiplicative depth of the circuit to be computed. Recall that each function can be implemented by a circuit of logical and gates (multiplication) and exclusive-or gates (addition). There are also an additional round for distributing the input shares and a round for combining the output shares.

The synchronization requirements are also met, i.e. the server only needs to wait for all clients to complete the previous. We can see that between round 1 and 2 and round 2 and 3 the messages sent are only dispatched to different recipients. Note that, even without the central coordinator no participant could begin a round before the previous has finished, since he requires the inputs of all other parties to proceed.

The BGW protocol is general, i.e. it can be used to implement any polynomial-time functionality, but even the centrally coordinated version cannot be used to substitute for our benchmarking protocol, since the clients are not anonymous amongst each other, but are addressed by their public keys.

A similar limitation as to the study applies to the generalization. It is unclear whether the computation performed outweighs the network delay. The clients have to perform a LaGrange interpolation during the combined round 1 and 3 and n modular multiplications during round 2 per multiplication gate. Several gates can be performed in parallel. In the benchmarking protocol the computational load is unbalanced between rounds. It ranges from n decryption operations and 3 OTs in round 2 to a computation of $n + 1$ cryptographic hash functions in round 5.

The benchmarking protocol can be completed in 5 rounds while the BGW protocol's number of rounds depends on the circuit. A simplified benchmarking circuit could be implemented using a circuit with depth $\log^2 n$ for a sorting network to compute the ranks of the input values plus some addition gates. Addition gates can be implemented with depth $\log x_i$, such that the number of rounds in the BGW protocol is larger, but by a low factor for real-world problem sizes.

5. RELATED WORK

We are aware of four scientific reports on practical implementations of SMC protocols [4, 5, 9, 17]. In [5] a multi-commodity auction protocol is implemented whose experiences from practical execution are reported in [4]. No evaluation of the performance characteristics is done except a general feasibility of the protocol given the performance for particular operations. In [9] a survey system is implemented resembling our benchmarking system. They implement a set of very similar statistics based on the implementation of [17], although they use a model with two servers similar to [20]. They also do not report detailed performance figures except general feasibility, but report that memory consumption for the garbled circuit was the bottleneck.

The FairPlay paper [17] presents the most general solution implementing the two-party protocol of [27]. The system contains a compiler translating a programming language into an executable circuit which can be run by the protocol. They ran their examples over a LAN and a real WAN connection and report that 57% to 86% of the overall time was spent due to network conditions. No attempts to optimize this figure are reported, since they focused on optimizing the OT performance which accounted for up to 91% of the overall performance. They also ran very small examples which ran on the order of seconds and not real-world problem sizes as ours which require minutes to complete.

Further practically implemented two-party protocols mostly based on Yao's garbled circuit protocol are presented in [6, 13]. In [6] a remote diagnostic system is implemented which runs on the order of a very few minutes (without network considerations) is described. In [13] the implementation for gene sequence alignment using edit distance or similar algorithms is described. The running time is on the order of minutes for already practical, albeit very small problems.

A similar study to ours considering the problem of private information retrieval (PIR), a two-party problem, has been done in [26]. Although only for a two-party problem, the authors reach a similar conclusion that given high computational cost network performance is negligible. We extend their work by adding a possible synchronization pattern for SMC.

6. CONCLUSIONS AND FUTURE WORK

We presented a benchmarking protocol for securely computing several statistics using an oblivious server. We showed that this protocol can be implemented, such that the network performance has almost no impact on the overall performance, if we follow a specific synchronization and communication pattern, i.e. the protocol runs as fast, if the clients are joint on a LAN, as if they are distributed over the Internet. We used real-world problem sizes under realistic network conditions in the evaluation. We then showed that the landmark protocol by Ben-Or, Goldreich and Wigderson can be implemented using the same pattern. We claim that this shows that secure multi-party computation (SMC) protocols can be implemented, such that the communication complexity for real-world applications under realistic network conditions is negligible for the overall practical performance, although some limitations to the generalization regarding the relationship of computation and network delay exist.

Round 1:			
X_i	$\longrightarrow$	SP	$E_1(h_{i,1}), \ldots, E_{i-1}(h_{i,i-1}), E_{i+1}(h_{i,i+1}), \ldots, E_n(h_{i,n})$
Round 2:			
SP	$\longrightarrow$	X_i	$E_i(h_{1,i}), \ldots, E_i(h_{i-1,i}), E_i(h_{i+1,i}), \ldots, E_i(h_{n,i})$
X_i	$\longrightarrow$	SP	$E_1(k_{i,1}), \ldots, E_{i-1}(k_{i,i-1}), E_{i+1}(k_{i,i+1}), \ldots, E_n(k_{i,n})$
Round 3:			
SP	$\longrightarrow$	X_i	$E_i(k_{1,i}), \ldots, E_i(k_{i-1,i}), E_i(k_{i+1,i}), \ldots, E_i(k_{n,i})$

Figure 7: BGW protocol with central coordination

We argue that more attention should be paid to improvements in computational complexity than improvements in communication complexity.

7. REFERENCES

[1] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. *Proceedings of the ACM Workshop on Privacy in an Electronic Society*, 2004.

[2] J. Benaloh. Verifiable Secret-Ballot Elections. *PhD thesis, Yale University*, 1987.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th ACM symposium on theory of computing*, 1988.

[4] P. Bogetoft, D. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Kroigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach and T. Toft. Multiparty Computation Goes Live. Available at *http://eprint.iacr.org/2008/068*, 2008.

[5] P. Bogetoft, I. Damgard, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. *Proceedings of Financial Cryptography*, 2006.

[6] J. Brickell, D. Porter, V. Shmatikov, E. Witchel. Privacy-Preserving Remote Diagnostics. *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[7] I. Damgard, and M. Jurik. A Generalisation, a Simplification and some Applications of Paillers Probabilistic Public-Key System. *Proceedings of International Conference on Theory and Practice of Public-Key Cryptography*, 2001.

[8] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM 28(6)*, 1985.

[9] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure Computation of Surveys. *Proceedings of the EU Workshop on Secure Multiparty Protocols*, 2004. Available at *http://www.cs.yale.edu/homes/jf/SMP2004.pdf*.

[10] O. Goldreich. Secure Multi-party Computation. Available at *www.wisdom.weizmann.ac.il/~oded/pp.html*, 2002.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th ACM conference on theory of computing*, 1987.

[12] S. Goldwasser. Multi party computations: past and present. *Proceedings of the 16th ACM symposium on principles of distributed computing*, 1997.

[13] S. Jha, L. Kruger, and V. Shmatikov. Towards Practical Privacy for Genomic Computation. *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[14] F. Kerschbaum. Practical Privacy-Preserving Benchmarking. *Proceedings of the 23rd IFIP International Information Security Conference*, 2008.

[15] F. Kerschbaum, and O. Terzidis. Filtering for Private Collaborative Benchmarking. *Proceedings of the International Conference on Emerging Trends in Information and Communication Security*, 2006.

[16] N. Lynch. Distributed Algorithms. *Morgan Kaufmann Publishers*, 1996.

[17] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System. *Proceedings of the USENIX security symposium*, 2004.

[18] D. Naccache, and J. Stern. A New Public-Key Cryptosystem Based on Higher Residues. *Proceedings of the ACM Conference on Computer and Communications Security*, 1998.

[19] M. Naor, and B. Pinkas. Efficient Oblivious Transfer Protocols. *Proceedings of the symposium on data structures and algorithms*, 2001.

[20] M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999.

[21] T. Okamoto, and S. Uchiyama. A new public-key cryptosystem as secure as factoring. *Proceedings of EUROCRYPT*, 1998.

[22] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of EUROCRYPT*, 1999.

[23] M. Rabin. How to exchange secrets by oblivious transfer. *Technical Memo TR–81, Aiken Computation Laboratory*, 1981.

[24] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review 27(1)*, 1997.

[25] A. Shamir. How to share a secret. *Communications of the ACM 22(11)*, 1979.

[26] R. Sion, B. Carbunar. On the Computational Practicality of Private Information Retrieval. em Proceedings of the Network and Distributed System Security Symposium, 2007.

[27] A. Yao. Protocols for Secure Computations. *Proceedings of the IEEE Symposium on foundations of computer science* 23, 1982.