Assisting Server for Secure Multi-Party Computation

Jens-Matthias Bohli, Wenting Li, Jan Seedorf

Assisting Server for Secure Multi-Party Computation

Jens-Matthias Bohli, Wenting Li, Jan Seedorf

NEC Laboratories Europe
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
firstname.lastname@neclab.eu

Abstract. Distributed threats like botnets are among the most serious threats in the Internet. Due to their distributed nature, these attacks are difficult to detect in an early stage without the collaboration of several network operators. However, the exchange of monitoring data between different parties turns out to be difficult in practice, due to the desire of operators not to disclose network internals and legal data protection requirements. Secure Multi-Party Computation (SMC) for privacy-preserving sharing of network monitoring data can be a solution to the problem. As real-time performance of SMC is important for this application, we investigate ways to speed up SMC.
The focus and contribution of our work is a new model for SMC that enables to increase the performance of certain SMC primitives significantly. We introduce an assisting server which operates on dedicated, intermediate data values in plaintext. The overall rationale behind our approach is that the performance gains outweigh the slight decrease in security introduced by revealing intermediate computation results to the assisting server. We propose a new primitive for checking the equality between two values, $equal^+$, based on our new model. Through prototypical implementation we compare $equal^+$ with existing algorithms. Further, we evaluate $equal^+$ in the context of a cooperative network monitoring application, link-counting. Our results demonstrate that certain SMC applications can be computed much faster with our approach. Finally, we discuss the security implications of the new model.

1 Introduction

In today's Internet, distributed threats including malicious cooperative attacks are a tough problem to solve. In principle, coordinated attacks could be identified if the communication graph, or social graph, between nodes can be analyzed as a whole [13, 7, 17]. However, the communication graph is distributed across multiple network operators and every network operator is monitoring their own network only. The analysis of the communication graph will therefore require collaboration between multiple operators and the sharing of monitoring data. This seems today to be neither likely nor desirable to happen. Besides the fact that operators tend to be reluctant to share detailed monitoring information due to their own secrecy requirements, such monitoring information might well

contain personal identifying information which is protected by data protection laws in Europe.

In this context, the EU project *DEMONS* [9] aims at providing a scalable and flexible monitoring infrastructure while ensuring user privacy of cross-domain cooperation. *Secure Multi-Party Computation (SMC)* is a cryptographic tool for privacy-preserving analysis of distributed data [19]. SMC takes private input data from multiple parties and carries out a joint computation on them, while ensuring that the input data remains private to their owners during the computation process. With the help of SMC, many cooperative computations among parties that do not necessarily trust each other can be achieved by keeping the privacy property of the data, thereby addressing a quite common problem in business and government administration.

Recently, SMC gets closer to practice. An example where SMC was applied in practice is a Danish auction of sugar beets [3]. Several frameworks for SMC have been implemented in the last years. SEPIA achieves the best performance among these frameworks and has been used for the analysis of network statistics [14]. Other frameworks are e.g. FairplayMP [1] and VIFF [6].

In this paper, we present a trade-off that increases the efficiency of SMC at the cost of a limited information leakage under the secret sharing scheme. The motivation is that to be able to detect and stop attacks before any damage is done, the solution must be extremely efficient to fulfill the real-time requirements. Specifically, we introduce a special entity, an *assisting server*, that can compute some operations on plaintext. We extend SMC based on Shamir's secret sharing; our solution is to be seen as a building block for applications implemented using such an SMC scheme.

In this new assisting server model, we realize an improved algorithm equal$^+$ for checking equality of two values. The equal$^+$ algorithm runs 20 times faster than the current algorithm for input data of 32 bits such as IP-addresses. We use this equal operation to realize a *link-counting* protocol which is an important building block in the analysis of shared monitoring data. Our equal$^+$protocol leaks a controlled small amount of information to the assisting server. We analyze the security setting where such an information leakage can be tolerated.

2 Preliminaries

2.1 Related Work

Secure computation can be realized in various ways. Some protocols are customized for specific operations, such as [16] to sum up the input data, or [18] to realize *xor*, scalar product of vectors, and equality comparison. The data in each individual operation is processed using different techniques such as encryption or random numbers. The information exchanged among communication parties also varies for each operation or scheme. Therefore, a major drawback of these standalone protocols is that they cannot be easily combined to solve a complicated function composed of different operations, as the output of each step needs transformation for further processing.

On the other hand, much work has been done to construct a generic scheme that processes the data in a uniform representation. Yao as well as Goldreich et al. introduced the idea of a *garbled circuit* [20, 12]. The function to be computed is first transformed to an encrypted circuit; then an evaluator obtains obliviously the keys for each bit of the input from the circuit creator and evaluates the circuit through partial decryption. In theory, any function can be computed in this scheme. The only concern is the performance, as public-key primitives are needed to retrieve every bit of the input. Another way for secure computation is the use of homomorphic encryption. Fully homomorphic schemes were recently proposed [11], however, the performance of these fully homomorphic schemes is currently far from being practical.

SMC based on *Shamir's Secret Sharing*, which we will build on, was introduced in [2]. Compared to the above two schemes, the distribution and reconstruction of the data is trivial. There is no need for oblivious transfer or encryption. Evaluating linear functions turns out to introduce almost no overhead. Its limitation, however, is the complexity when dealing with non-linear functions and boolean operations. Many rounds of communication among the parties that hold the shares are required to achieve such functions with state-of-the-art algorithms. Our objective is hence to improve the performance of such functions, towards using secret sharing as an efficient generic scheme for any type of computation.

We consider in this work security/efficiency trade-offs and allow a limited information leakage to an assisting server. This distinguishes the assisting server from an oblivious third party that does not gain any knowledge as used in [4] The idea of the assisting server is therefore closest to the *untrusted third party* in [18]. However, the untrusted third party is used for simple operations as a stand-alone protocol. Our assisting server operation generalizes and improves the concept towards arbitrary functions through the integration in SMC computations. Further, in [18] an equal protocol is presented based on asymmetric encryption. The equal function we propose is much more efficient as it just needs multiplication with a random number.

2.2 SMC with Shamir's Secret Sharing

An (n, k) secret sharing allows one party to distribute a secret among a set of n parties in the form of shares, such that a specified subset of the parties (k out of n) can reconstruct the secret while ensuring that less than k shares can infer no information about the secret. Shamir's secret sharing relies on the idea that it takes at least k points to uniquely define a polynomial $p(x)$ of degree $(k-1)$. To generate shares, a random polynomial $p(x)$ is first constructed by choosing $k-1$ random numbers $a_1, \cdots, a_{k-1}$ as the coefficients and the secret s as the free coefficient:

$$p_s(x) = s + a_1 x + \cdots + a_{k-1} x^{k-1}.$$

For SMC, we require $n > 2k$ parties to hold the shares. Every party gets assigned a coordinate x_i. We denote by $\{s\}_i = p_s(x_i)$ the share of party i. By combining

any k (or more) shares together, we can rebuild the polynomial and recover the secret using Lagrange interpolation. Therefore, the distribution and reconstruction of a secret only need arithmetic evaluation, which is trivial comparing to encryption and decryption operations.

Addition in Shamir's secret sharing scheme is straightforward as it is a linear scheme. Given two secrets a and b shared by polynomials of degree $d = k - 1$, $p_a(x)$ and $p_b(x)$, the sum s is given by the polynomial $p_s(x) = p_a(x) + p_b(x)$ as $p_s(0) = p_a(0) + p_b(0) = a + b$. Therefore, the new shares of the sum can be computed by simply adding up the corresponding shares of each party: $\{s\}_i = \{a\}_i + \{b\}_i$.

The multiplication of two polynomials $p_s(x) = p_a(x)p_b(x)$, however, doubles the polynomial degree. It means that although the product result is correct: $p_s(0) = p_a(0)p_b(0) = ab$, it requires $2d + 1$ shares to reconstruct the polynomial this time. Therefore, we need $n > 2d$ parties to reduce the polynomial degree as well as restore the randomness of the coefficients, which introduces communication among the parties as a result. The protocol we adopted is described in [10] and needs one communication round to redistribute the shares after a multiplication.

Other primitives such as comparison, are mainly designed based on an algorithm build on additions and multiplications [14]. The algorithms are more complex as they invoke heavy communication among parties for such operations as multiplication or shared random number generation. Meanwhile, they should also involve as few communication rounds as possible, as each round will introduce network overhead. Therefore, we evaluate the performance of each primitive algorithm mainly according to two criteria. One is the number of multiplications it invokes, which relates to the amount of message exchanged among parties as well as local computation time[1]. The other criteria is the number of communication rounds. All messages to be exchanged are collected until no more computations can be executed. Then in one communication round, all buffered messages are exchanged. Thus, the message sent in a communication round can include the shares of several multiplications done in parallel. The efficiency of a communication round depends on the latency of the network used. For efficient protocols, in terms of computation and communication, both the multiplications and communication rounds should be minimized. The assisting server described in Section 3 aims at decreasing the number of multiplications and communication rounds in SMC.

2.3 Adversary Models

Two adversary models are commonly considered for SMC, depending on the kind of control that the adversary has over the corrupted party:

- *Semi-honest model:* The adversary has the ability to collect and read the information of a corrupted party, but still executes the protocol correctly. In

[1] Any operation that requires a communication round afterwards, will count as one multiplication"

other words, the adversary only intends to break the privacy of the protocol. We call this type of adversary *honest-but-curious adversary* or *passive adversary*
- *Malicious model:* The adversary takes full control of a corrupted party. Thus, the party can actively launch an attack inside the protocol and break the correctness of the protocol. We call this an *active adversary*.

The privacy requirement of an SMC protocol is that no one can learn any information about the input data out of his own possession during the computation process, except for what can be inferred from the output of the function. Regarding to SMC based on secret sharing, as long as the data always remain in the form of shares and not enough peers are colluded, the privacy of the input data can be ensured.

For our SMC application (cooperative network monitoring), the semi-honest model is reasonable: The various network operators involved are actually interested in the result of the joint computation and the prevention of attacks to their networks. Therefore, we assume that all the parties will provide the correct data and follow the protocol honestly. Meanwhile, they will not take the initiative to collude with other parties, as none of the operators would risk leaking their own private data in exchange of those of the others. Moreover, as Shamir's secret sharing is a threshold scheme, even when some parties are compromised by external adversaries and collude with each other, the scheme can still resist the collusion of a certain number of corrupted parties. In fact, out of n shares of a k-degree polynomial, the secret remains confidential as long as no more than $k+1$ shares are combined. Therefore, it is reasonable to assume that the parties only behave passively and try to infer as much information as they can from the data they obtained during the computation.

3 Assisting Server for Secure Multi-Party Computation

This section introduces the general concept of the *assisting server*, followed by the example of equal$^+$ using an assisting server. The analysis will show an improvement in terms of computation and communication rounds.

3.1 Assisting Server Model

Traditionally, SMC is done in alternate computation and communication rounds. In the computation rounds, each party computes locally on shares it holds. In the communication rounds, the parties create new shares that they distribute to other parties. With an assisting server, a new type of round is possible. In such a round, the parties communicate with the assisting server in the following way:

- The parties send a message to the assisting server. This message triggers the service requested from the assisting server, but might also include shares or plaintext information that can be reconstructed and used by the assisting server.

– The assisting server responds to the parties of the SMC. Usually, this response includes shares that the parties will subsequently use in their computation. But the response may also consist of plaintext information.

The intended efficiency gain can be achieved, if one round of communication with the assisting server can replace multiple rounds of communication between the SMC parties in the traditional model. The efficiency gains that can be achieved in the assisting server model come at a certain cost in security. The assisting server gains certain knowledge of the computation that is ongoing. Which information is revealed depends on the algorithm that is computed. We give more details in the description of the protocols for equal$^+$, and a short security discussion in the application scenarios in Section 5. As the leaked information and the impact of the leaked information is highly application dependent, setting up a SMC and choosing the right protocols becomes more difficult than with traditional multiparty computation which does not leak information. However, our use-case shows that the efficiency gain can be essential and makes it worthwhile to search for application-specific improvements.

The assisting server offers its service in a stateless way. The parties can ad hoc choose any assisting server that offers the necessary functionality, if it is required in the computation process. In case no additional information can be gained by a collusion between the assisting server and a party, one of the party could play the role of an assisting server by itself. The security assumptions concerning the assisting server are similar to the existing security assumptions. We assume that the assisting server behaves like a passive adversary. It does follow the protocol and does not collude with one of the parties in order to gain or share information. The assisting server may be interested and store all information it learns during the computation. Our requirement is not to prevent the assisting server to learn any information, but allow that a limited well-defined amount of information can be learnt. This definition needs to be evaluated and defined anew for each application where this approach is used.

3.2 Equality Comparison *Equal*$^+$

Comparing equality between two input data is one of the basic operations that SMC should provide and is used in many applications to match features in different data sets, such as evaluating conditions in privacy-preserving data mining, or aggregating data provided by different providers. The current algorithms for equality comparison under secret sharing scheme can be found in [5], [15], [14]. The protocol of [5] performs a bit-decomposition of the secret shares which is an expensive operation as well. A constant round protocol is given in [15] but requires $98l + 94l \log_2 l$ multiplications, where l denotes the bit length of p in field $\mathbb{Z}_p$ ($l = \lfloor \log_2 p \rfloor$). SEPIA [14] focuses on efficiency and aims not at constant-round communication. It adopts Fermat's little theorem (see Equation 1) and invokes $l + k - 2$ multiplications and l rounds, where k denotes the number of bits set to 1 in $p - 1$. The algorithm is

$$equal(\{a\}, \{b\}) = \{1 - (a - b)^{p-1}\}. \tag{1}$$

The efficiency of equality comparison is still not suitable for real-time applications, as l can easily exceed 32.

Therefore, we propose a probabilistic protocol referred to as $equal^+$ with the help of a shared random number and an assisting server for intermediate result evaluation.

Equal$^+$. At first, an unknown random number will be shared among the parties as $\{r\}$ using the *Joint Random Number Sharing protocol* from [15]. This is done by each party generating a local random number and distributing its shares to other parties. Then each party adds up the obtained and generated shares and the result is his share of the joint random number r. The random number r can be seen as an oblivious random number uniformly distributed over $\mathbb{Z}_p$. Its value cannot be predicted or influenced as it depends on the local random numbers generated on all parties, unless they are all colluded.

Then shares $\{c\}$ of $c = r(a-b)$ are computed by the parties by first subtracting the shares of b from the shares of a and the multiplying the shares of r and $a-b$. As we can see, c will be 0 if and only if r or $a-b$ equals to zero. With a sufficient big field size, which is usually the case, the probability $Prob(r = 0) = \frac{1}{p}$ which is negligible. Therefore, we can say that with overwhelming probability $c = 0 \Leftrightarrow a = b$.

Given that the value of r is randomly distributed, c can also be seen as a random number if $a \neq b$. Thus, c can be revealed without disclosing anything about a and b apart from $a = b$. The parties send the shares of c to the assisting server. The assisting server reconstructs c and evaluates if $c = 0$. As a result, in our solution the assisting server only has the knowledge of the equality comparison result. Nevertheless, we still need to analyze if this kind of information leakage is secure to a specific application.

What is left to complete in this function is mapping the *zero-/non-zero-*result to a binary value and sharing it back to the parties. This is done by the exponentiation using Fermat's little theorem in the previous protocol and will be done by the assisting server in our case. Therefore, our optimized equality comparison is defined as

$$equal^+(\{a\}, \{b\}) = \text{isZero}(\text{recon}(\{r \times (a-b)\})), \tag{2}$$

where recon$(\cdot)$ denotes the reconstruction function of a secret, and isZero$(\cdot)$ is the evaluation function that maps 0 to 1 and any value $\neq 0$ to 0.

The execution of $equal^+$ in (2) takes four rounds: one for receiving the shares of r, one for the multiplication result, one for sending the message to the assisting server, and one for receiving the answer from the assisting server. Comparing to SEPIA's version with $l + k - 2$ multiplications in l rounds, our solution is more efficient, with a constant and small number of computation and communication rounds.

3.3 Applications for Cooperative Network Monitoring

We present an application protocol called *link-counting* that can be used for privacy-preserving, cooperative analysis of monitoring data between different

operators. This protocol provides a database-query-like operation with two input parties. It sums up the number of matched records with specific conditions. It can for instance be used in distributed monitoring systems for Spam detection based on social graph analysis. In principle, the protocol enables two or more parties to compare and match social graphs without revealing the internal links among nodes within each network to each other.

The concept of link-counting is very similar to the *Database Query* problem described in Du's SMC problem definition [8]: Domain A has the identities of a list of users $Q = (q_1, \cdots, q_M)$, and domain B has a database of user linking records $(U, V) = \{(u_1, v_1), \cdots, (u_N, v_N)\}$; A wants to know how many records of (u_i, v_i) in B's database match the combination of (q_r, q_s), $i = 1, \cdots, N, r, s = 1, \cdots, M, r \neq s$. The privacy requirement is that B cannot know A's secret query Q or the response to that query, and A cannot know B's database contents except for what could be derived from the query result. The implemented algorithm checks for each edge (u_i, v_i), if $u_i \in Q$ AND $v_i \in Q$ and sums up the resulting values. This algorithm requires $2MN$ equality tests and N multiplications. We will see in Section 4.1 that the performance of this protocol is more acceptable when using equal$^+$to perform the equality test.

4 Implementation and Evaluation

In this section, we present the experimental results of our performance evaluation on equal$^+$using an assisting server as described in Section 3.2. The objective of the experimental tests is to show the consistency with our theoretical analysis and the performance advantage of our improved SMC primitive. Further, we study the running time of the link-counting protocol. The experiments will be in comparison with the *equal* algorithm in [14] because this shows the best performance among SMC frameworks and includes a comparison with the other frameworks.

We implemented our algorithms in the SEPIA framework, a Java library providing the tools for privacy-preserving aggregation of multi-domain network data [14]. The SEPIA framework adopts Shamir's Secret Sharing scheme (see Section 2.2). There are two roles in the framework, the *input peer* and the *privacy peer*. Input peers are the parties who provide private data for the joint computation. At the beginning of the computation, the input peers distribute the shares of their data among the privacy peers using Shamir's secret sharing. Privacy peers (PPs) hold the shares of the data and perform the computation on them. They will not reconstruct the inputs nor the intermediate results; they only reconstruct the final result and send it back to the input peers.

The threshold t is chosen as half of the parties: $t = \lfloor (n-1)/2 \rfloor$. Thus, the minimum number of parties is 3 when $t = 2$. With more parties, the performance will decrease as it will involve more message exchanges and synchronizations among the parties. The semi-honest adversary model suggested in SEPIA (see Section 2.3) is quite reasonable in a real application: Different business parties usually do not intend to break the protocol, but they are still interested in

inferring as much information as they can during the computation to know the private data of the other parties.

4.1 *Equal*$^+$ Performance Test

The test environment of our experiments is on 3 separate PCs (Athlon XP 2800+ or better) and each runs a privacy peer process on it. The other peers including input peers and assisting peers are distributed evenly on the same group of machines. For each test, the average of 100 program executions is taken. The absolute number of equal operations per second using *equal* differs from the result in [14] because of the different test environment.

In the first experiment, we want to show that unlike *equal*, the performance of *equal*$^+$ is hardly influenced by the value of the polynomial field size p. The result confirms our theoretical analysis on the computational complexity in Section 3.2. The second experiment is a benchmark test on how many equal operations can be accomplished using each tool. The last experiment compares the performance of the link-counting protocol. The result shows that the choice of primitive operations in an application determines if the running time is acceptable or not, and that using *equal*$^+$ is a much more reasonable choice for a real-world application.

Experiment 1. Different Field Size In Section 3.2, we compared the complexity of *equal* using Fermat's little theorem and *equal*$^+$ using an assisting server by analyzing the number of multiplications and communication rounds. Our analysis shows that *equal* needs $l+k-2$ multiplications and l communication rounds, where l is the bit length and k the hamming weight of the prime p. In comparison, *equal*$^+$ only requires 4 multiplications and 4 communication rounds. As a result, we can conclude that the performance of *equal* depends on the selection of the size of $\mathbb{Z}_p$, which is chosen depending on the values in the computation because any input/output or intermediate data should be smaller than p. To optimize the performance of *equal*, we choose a prime p with $k \leq 3$ which is the optimal choice to reduce the complexity.

Figure 1 shows the results of the run-time of each equal operation in relation to the size of the data space. In the test, the prime p that determines the data space is selected as the smallest prime that can cover the data space and has a weight $k \leq 3$. As expected, with increasing data space size, *equal* takes more time to accomplish one operation, while the performance of *equal*$^+$ keeps quite stable under different application data space. The sudden increase for both operations at a data space size of 32 bits is caused by the implementation of the SEPIA library. When dealing with modular multiplication, the program changes from a primitive *long* datatype ($-2^{63} \sim 2^{63}$) to a more complex data structure *BigInteger*. The impact of changing the Java data type is smaller on the performance of *equal*$^+$ because *equal*$^+$ uses less modular multiplications.

Figure 2 shows another perspective of the performance comparison. Since in principle multiplication and communication rounds are the only factors of the computational complexity, *equal*$^+$ should perform between $l/4$ and $l/3$ times faster than *equal*. We use the results from Figure 1 to plot the ratio of *equal*$^+$

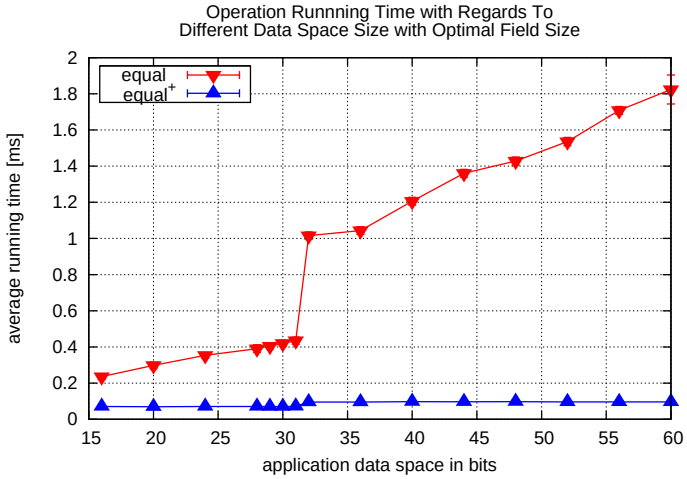

Fig. 1. Average running time of an equal operation over different application data space. The field size for each data space is selected as the smallest applicable prime number that has $k \leq 3$.

over *equal* on different values of l. Meanwhile, we also display the ideal performance ratio $l/4$ and $l/3$ (in dashed and dotted lines). We can see that most part of the experimental curve is within $l/4$ and $l/3$, which is consistent with our assumption.

Experiment 2. Benchmark for Equality Comparison. In this experiment, we fix the field size and test how many equal operations each algorithm can accomplish in one second (equals/s). Instead of giving a single result, we change some test conditions for benchmarking and discuss the outcome. In the configuration, we decide how many equal operations to run in a round by setting the test set size. We then execute the test and measure the completion time. The total number of equal operations (or the test set size) divided by the completion time is our benchmark. We run the test with different test set sizes and with different application data space sizes. The result of the benchmark is shown in Figure 3. As can be seen from the results, *equal*$^+$ is able to do 10-20 times more equal operations than *equal* when processing 32-bit integers, and 20-40 times more operations with 60-bit integers.

We notice that the efficiency of our equal operation increases with the test set size. The reason is that SEPIA executes the operations in parallel thus the number of communication rounds does not increase but the CPU utilization between rounds is higher. In addition, we observe in Figure 3 that both equal algorithms perform less efficiently with 60-bit integers than with 32-bit integers. This is within our expectation because computations over larger values are usually slower. However, while the performance of *equal*$^+$ only decreases by 1.64% because of the larger data to process, the benchmark of *equal* reduces by 42.44%.

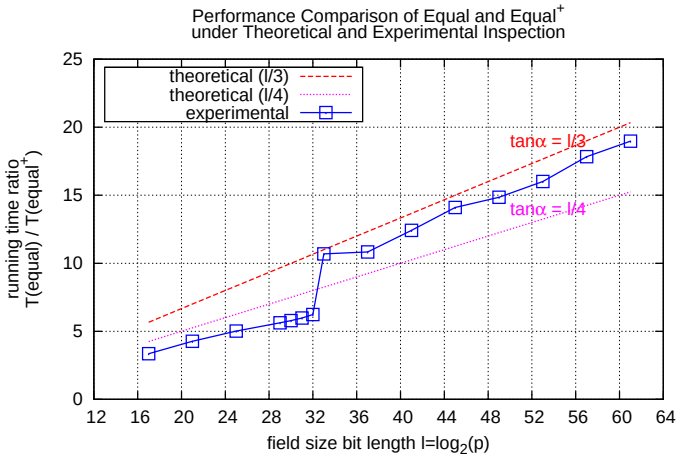

Fig. 2. Running time quotient of *equal* over $equal^+$. The non-solid lines indicate the theoretical values from the computational complexity analysis.

This confirms that the field size is a decisive factor for the performance of *equal*. Therefore, we can predict that if an SMC application deals with bigger data space, the advantage of $equal^+$ will become more obvious.

Experiment 3. Performance in Applications. We test the performance of cooperative network monitoring using *equal* and $equal^+$, respectively. We are interested to verify that link-counting will be more practical in terms of execution time with the optimized primitive $equal^+$. For the link-counting protocol, the main performance challenge is the potentially large amount of records in the database. For network monitoring, a database B contains all the links between users in the network, which can easily be over a million records. However, in a real application, it is not necessary that party B provides all records to an SMC function. In fact, the query party A can provide some information regarding the range of the queried users to decrease the search space.

In the performance test, we fix the query size to 10 users, and incrementally increase the number of records in the database from 10,000 to 100,000. Figure 4 shows the performance of two implementations of the link-counting protocol, one based on *equal* and one based on $equal^+$. It can be observed that using $equal^+$ on 100,000 records, the query process takes less than 3 minutes, while *equal* needs to run more than half an hour.

4.2 Evaluation Summary

We used three commodity PCs in our testing environment. In a real application scenario, the absolute execution times shown in our results can be improved by using powerful machines to run the protocols. Moreover, since $equal^+$ involves

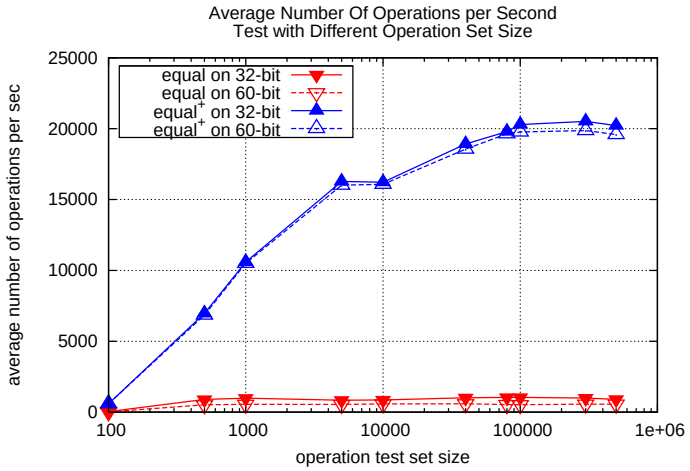

Fig. 3. Benchmark of *equal* and $equal^+$ on number of operations per second. The benchmark runs under different test conditions: various data space size (32-bit and 60-bit) and various operation test set size.

less communication rounds, the protocol will suffer less from a distributed and asynchronous network, and thus, profit more from powerful CPUs.

The experimental results of the algorithm $equal^+$ using an assisting server provide evidence of higher efficiency and stability compared to existing algorithms. The field size test shows that $equal^+$ performs stable over different application data space. The benchmark test provides an estimate of how many equality comparisons both algorithms can accomplish in a unit of time. The result data shows that $equal^+$ runs 10-20 times faster than the previous algorithm when dealing with 32-bit integers. The benchmark test on different operation settings also show that - due to the parallel execution feature in SEPIA - the protocols perform more efficiently when processing data in a batch job. This is an advantage when dealing with big data sets, but we need to choose carefully how big the parallel operation set is in order to yield optimal performance.

The application protocol link-counting has been developed with cooperative network monitoring in mind. In the experimental tests of this protocol we only used data sets of medium size, e.g. the maximum size of the database is 100,000 records. Further, because SEPIA executes the operations in parallel, it will pack all operational data in a single message, which can become very large when the protocol executes a lot of operations. Nevertheless, the performance results of our protocols show that $equal^+$ is in principle applicable on big data set operations.

In summary, our results provide evidence that the assisting server model in conjunction with the new $equal^+$ is more efficient in execution time than existing algorithms. It is a promising solution to provide practical protocols for performance-demanding applications. We gave examples how the implementation of a protocol also affects application performance.

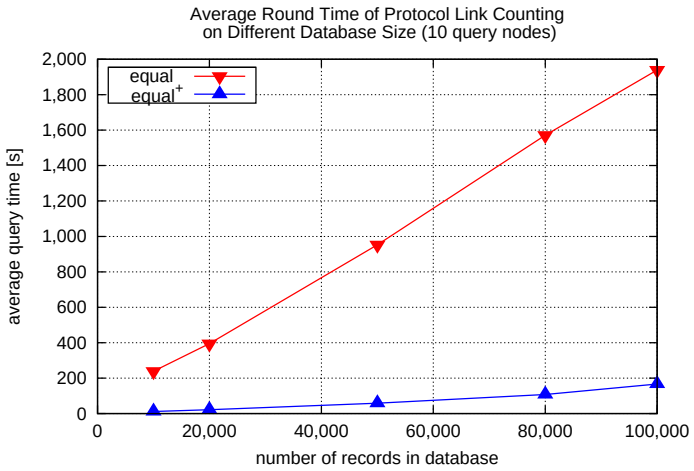

Fig. 4. Performance of link-counting protocol using *equal* and $equal^+$ test on different size of database.

5 Security Discussion on the new SMC Model

We have shown that using an assisting server can make certain SMC functions much faster; we will now discuss the security implications introduced by our new model. For the parties, the security assumptions remain. In the semi-honest model, the resistant threshold for collusion still remains $n/2$, where n is the number of parties. A collusion involving at most half of the parties cannot extract any information from the computation. Certain assumptions are necessary for the assisting server: It is assumed that the assisting server does not collude with the parties and will correctly follow the protocol. The assisting server might however be curious and gather as much information as possible. The assisting server is explicitly allowed to gain some information about intermediate results of specific operations. In the case of equal$^+$, this is the result of the equality operation. The assumptions for the assisting server are clearly weaker than the assumptions for a trusted third party (TTP). A TTP would be allowed to know all the private input data, while the assisting server is only trusted to know well defined partial information of some intermediate computation steps. The limited amount of information prevents the assisting server for instance from doing large scale data mining. third party.

Further strategies are possible to mitigate the risk of the information available to the assisting server: 1) Using an independent assisting server offering the service publicly. The server might have no context information about the origin of the evaluated data. Therefore, the assisting server will not have a relevant knowledge gain; 2) Preprocessing of the data by the parties, e.g. shuffling with the help of a different assisting server, might again help to reduce the information gained by the assisting server, under the assumption that the two assisting servers do not collude.

The assisting server model demands an application-specific security analysis. In each application using an assisting server, we need to identify whether the information leakage is acceptable by the application. Also what kind of information can be inferred through a collusion of a party with the assisting server is relevant for assessing the risk.

In many cases, the information learnt by the assisting server might be information that will finally be public anyway because it is part of the output of the parties. In that case, the intermediate results obtained by the assisting server do not give any new information even when shared with a party. In that case it is possible that one of the SMC parties plays additionally the role of the assisting server for certain functions. This scenario is in general more challenging as it corresponds to the situation of a collusion between the assisting server and a party.

Link-counting. In the link-counting protocol using $equal^+$, the assisting server learns the results of the equality comparison. The order of equal or not-equal results might leak some data about the structure of A's request and B's network that should not be disclosed to an outsider. This could be mitigated by the involved parties, e.g. by agreeing on a different random program-flow, so that the order of execution is not related to the order of elements in the input vectors.

In case one of the parties takes the role of the assisting server this is not enough. However, if the program flow is randomized in a way oblivious to the parties, e.g. by a *secret shuffle* function, this approach is still possible. Given a set of shares $\{\{a_1\}, \ldots, \{a_m\}\}$, a shuffle function produces a set of shares $\{\{b_1\}, \ldots, \{b_m\}\}$ such that $a_i = b_{\pi(i)}$ for a random permutation π. The shuffle function needs to be secret, such that no party learns the new sequence order except the one that defines the shuffle function. After the operation, even if a party knows one of the input data sets and the output of a function, it cannot be sure which of the input items corresponds to the output of the function. Therefore, the secret shuffle adds uncertainty to an SMC function, hence reducing what an adversary can learn by colluding the party with the assisting server to obtain both the input and output of a function.

Secret shuffle using an assisting server. The most straightforward way of permuting a vector is by multiplying the vector with a permutation matrix. A permutation matrix is a binary matrix with exactly one entry 1 in each column and row. If this is done by a multiplication on shares of the vector and shares of the matrix entries, the re-randomization from the share-multiplication brings the secrecy requirement of the shuffle for free. A shuffle matrix could efficiently be created and distributed by an assisting server that is not involved in any further processing steps.

6 Conclusion

We have proposed a novel model for Secure Multi-Party Computation with a new entity, an *assisting server*. Based on this model, we introduced new algorithms for

the SMC primitive equality (called $equal^+$), as well as one application protocol for cooperative network monitoring, the *link-counting* protocol.

Our experimental evaluation with a prototype implementation in the SEPIA framework demonstrate that $equal^+$ can be computed much faster than previous algorithms in the traditional SMC model. Also, we showed that $equal^+$ significantly increases the performance of the link-counting protocol, and that its performance is less dependent on certain parameters than the original equal function in SEPIA. The performance gains come at the price of a certain relaxation of the security assumptions for SMC: the assisting server learns some intermediate computation results in plaintext. We have discussed options for mitigating this issue and provided concrete examples for preventing the assisting server from gaining too much insight into the overall SMC computation. Overall, we believe that algorithms can be designed in such a way that disclosing intermediate values to an assisting server is acceptable from the security perspective. In that case, the benefits of our approach, i.e. performance gains, clearly outweigh its drawbacks.

As future work, we intend to investigate other SMC primitives which can be sped up with an assisting server and to study the corresponding security implications formally.

Acknowledgment This work was partially supported by DEMONS, a research project supported by the European Commission under its 7th Framework Program (contract no. 257315). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DEMONS project or the European Commission.

References

1. 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 '08, pages 257–266. ACM, 2008.
2. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88*, pages 1–10. ACM, 1988.
3. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security, FC'09*, pages 325–343. Springer, 2009.
4. Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM conference on Computer and communications security, CCS'99*, pages 120–127. ACM, 1999.
5. Ivan Damgrd, Matthias Fitzi, Eike Kiltz, Jesper Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer Berlin / Heidelberg, 2006.

6. Ivan Damgrd, Martin Geisler, Mikkel Krigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *PKC*, pages 160–179, 2009.
7. Prasanna Desikan and Jaideep Srivastava. Analyzing network traffic to detect e-mail spamming machines. In *Proceedings of the 2004 ICDM Workshop on Privacy and Security Aspects of Data Mining (PSDM'04)*, 2004.
8. Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *In New Security Paradigms Workshop*, pages 11–20, 2001.
9. FP7-DEMONS.eu. Demons: Decentralized, cooperative, and privacy-preserving monitoring for trustworthiness. http://fp7-demons.eu/.
10. Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 101–111, 1998.
11. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.
12. Oded Goldreich, Silvio M. Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM Symposium on Theory of Computation*, STOC '87, pages 218–229. ACM, 1987.
13. Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, pages 139–154. USENIX Association, 2008.
14. Burkhart Martin, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security SYMPOSIUM*. USENIX, 2010.
15. Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *Proceedings of the 10th International Conference on Practice and Theory in Public-key Cryptography*, PKC'07, pages 343–360. Springer-Verlag, 2007.
16. Matthew Roughan and Yin Zhang. Privacy-preserving performance measurements. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, MineNet '06, pages 329–334. ACM, 2006.
17. Dominik Schatzmann, Martin Burkhart, and Thrasyvoulos Spyropoulos. Inferring spammers in the network core. In *Passive and Active Network Measurement*, volume 5448 of *Lecture Notes in Computer Science*, pages 229–238. Springer, 2009.
18. Jaideep Vaidya and Chris Clifton. Leveraging the "multi" in secure multi-party computation. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03, pages 53–59. ACM, 2003.
19. Andrew C. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.
20. Andrew C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167, 1986.