

Adapting Privacy-Preserving Computation to the Service Provider Model

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

Abstract

There are many applications for Secure Multi-Party Computation (SMC), but practical adoption is still an issue. One reason is that the business model of the application does not match the system architecture of regular secure computation. An important business model is that of a single service provider dealing with many customers. Applications with this business model are e.g. auctions or benchmarking. This paper provides SMC in a system architecture for service providers.

Furthermore we achieve an interesting performance improvement. Our SMC protocol has a significantly improved complexity if all parties behave semi-honest, but can still deal with a minority of malicious parties at the usual complexity.

The solution also relates to distributed algorithmic mechanism design which proposes to build distributed algorithms that implement mechanisms that compute the result given rational players.

1. Introduction

Secure multiparty computation (SMC) are cryptographic protocols that distributedly compute functions, such that the input of the parties remains private (i.e. confidential to that party). Only the result (or results) are revealed to the parties and what can be inferred from (one party's) input and output is implicitly revealed. Traditional SMC assumes that a subset (usually a majority) of the parties is honest and completes the protocol whereas the other parties are malicious and eventually may drop out of the protocol.

Most existing SMC protocols assume pair-wise secure channels between all parties. In contrast popular e-commerce sites, such as eBay or Amazon, follow a client-server model. The only secure channels are established between a single central server and a client. There are no channels among the clients and clients may even remain anonymous amongst each other.

Our SMC protocol uses such a central server that computes the result of the function and distributes it to the parties. It transfers constant ($O(1)$) communication units per link for most practical problems (circuit size $O(n)$) over a linear number of links in the number of participants ($O(n)$), if all participants behave semi-honest, and is therefore asymptotically optimal in this case. In case a $O(n)$ minority (e.g. $t < \frac{n}{2}$) behaves maliciously the communication complexity increases to $O(\alpha n)$ where α is the size of the circuit.

The introduction of a server does not trivialize the problem as in the ideal model of SMC. In the proof of our protocols the server acts as another player with no input, but receives the common output from the ideal trusted third party. The server facilitates the computation and significantly reduces complexity. Note also that we later proof security of the protocol against clients and server in the semi-honest and malicious model simply assuming there is no collusion, i.e. we are secure in the semi-honest and malicious model in any case, but performance increases if all parties behave honestly.

Our optimistic performance is an advantage to other theoretical solutions, that always transfer at least ($O(\alpha)$) communication units over linearly or even quadratically many links, e.g. [2]. All practical implementations of SMC [3], [10] have the same network complexity as our solution between clients and server, but they have multiple servers which then run a protocol with higher network complexity. While we significantly reduce complexity, a security drawback of our protocol is that we need to exclude collusion with this server. If the server “loses its status” and is seen as just another party in the protocol without input, it has like all practically implemented protocols [3], [10] a constant threshold against collusion.

We believe that the network complexity (along with the economic motivation) makes our protocols a good candidate for practical implementation. In summary, our protocol provides SMC for n parties using a

service provider, such that

- the protocol is “network efficient”, if all parties behave semi-honest.
- the protocol can be completed in a fixed, constant number of rounds as opposed to without a service provider [13].
- the clients may remain anonymous amongst each other.
- the computation requires only one server, as opposed to two [21].

The remainder of the paper is structured as follows: The next section reviews related work for efficient SMC, SMC using server models and rational SMC. Section 3 briefly describes Yao’s Garbled Circuits. Section 4 describes our server-assisted SMC protocol with its two sub-protocols. In Section 5 we analyze the security, rational behaviour and anonymity of the protocol. Section 6 presents the conclusions from our work.

2. Related Work

2.1. Efficient SMC

Many research results have focused on improving the performance of SMC, e.g. [2], [5], [6], [15], [16]. The best known solutions have (an abbreviated) communication complexity of $O(\alpha n)$ where α is the size of the circuit. Our server-assisted protocol achieves (an abbreviated) communication complexity of $O(\alpha)$, but this significant improvement comes at a cost. If the server is treated just as another party our protocol only provides security without collusion, i.e. while the other solution also aiming for efficiency are still resistant to collusion of up to at least $t < \frac{n}{3}$ parties, our threshold is $t < 2$. Our protocols are optimistic, such that they have a better communication complexity, if all parties behave semi-honest. They significantly improve performance over previous protocols in this best case and are the first “network efficient” SMC protocol with linear communication cost in the number of players (assuming a linear size circuit). The size of any useful n -party circuit is lower bound by $O(n)$, since n at least constant inputs are involved in the computation. Table 1 attempts a comparison overview.

This comparison is somewhat artificial, since none of the other efficient solutions focus on server-assisted protocols, but rather use a full mesh communication pattern. Hence, they do not offer the opportunity for anonymity as our protocol does. Adapting them to a server-assisted protocol is not always obvious.

In summary, our protocol significantly improves performance over previous protocols at the expense of

resistance against collusion and is the first “network efficient” SMC protocol with linear communication cost in the number of players in the best case.

2.2. Secure Computation using Server Models

The server model for general secure computation was introduced in [21]. It requires two mutually distrustful servers and the clients submit their inputs to one server only using a special proxy Oblivious Transfer protocol. A flaw in this protocol was later fixed by [18]. The overall protocol also used Yao’s Garbled Circuits where one server creates and encrypts the circuit and the other evaluates it.

Special protocols for auctions have been developed that use only one central server [4], [7], [8], [14]. These protocols only compute special functions necessary for the important problem of auctions (usually maximum). Their advantage is that they only require one server.

Multiple servers lead to different business models for the service provider of a privacy-preserving service. The service provider has to share benefits with an almost equal peer offering his computational power. In the one server model the service provider can offer the service by himself. In practice it can be very difficult for one to verify that two servers are really organizationally separated, although one can imagine that special service providers emerge that only offer their computation power for privacy-preserving services. For auctions the argument that auctioneer and seller provide two separate parties can be made [21], but this does not hold for all auctions either. E.g. the reverse auctioning platforms of large e-business software combines seller and auctioneer.

All implementations of SMC [3], [10] have adopted the server model, albeit with multiple servers. They argue for its scalability to many clients and account for the high traffic volume between server by provisioning adequate resources.

2.3. Rational SMC

Feigenbaum and Shenker review DAMD in [11]. It is the design of distributed algorithms that implement mechanisms such that a desired goal is achieved if all players act rationally. The designed algorithm not only serves its computational goals, but also works as a mechanism to facilitate the computation. The development of such algorithmic mechanisms was deemed necessary due to the observation that in many cases computations can be interrupted by misbehaving clients.

| Reference | Best-Case Communication | Worst-Case Communication | Collusion |
|------------|--|--|-------------------|
| [5] | $\alpha\beta n$ | $\alpha\beta n$ | $t < \frac{n}{2}$ |
| [16] | $\alpha n + \beta n$ | $\alpha n + \beta n$ | $t < \frac{n}{2}$ |
| [6] | $\alpha n + \delta n^2 + \text{poly}(n)$ | $\alpha n + \delta n^2 + \text{poly}(n)$ | $t < \frac{n}{3}$ |
| this paper | $\alpha + n$ | αn | $t < 2$ |

Table 1. Comparison of Efficient Secure Multi-Party Computation Protocols. α is the size of the circuit, β is the communication complexity of the broadcast primitive and δ the multiplicative depth of the circuit. Field sizes used for computation in the circuit and security parameters are assumed to be constant.

In many situations of the current Internet economy problems arise where cooperation is necessary, but often personal goals oppose the globally optimal solution. Routing in the Internet, in particular for multicast transmissions, is such a problem. DAMD was introduced for this problem in [9]. In [11] an overview is given and open problems are discussed.

One such open problem is to find SMC protocols that work when all agents are rational, i.e. neither honest nor malicious, which have low “network complexity”. (Rigorously formalizing “network complexity” is another open problem.) The first surprising result for this open problem was presented by Halpern and Teague [13]. They proved that no fixed round protocol can exist, if the players act rationally. They limit rational behaviour to players who want to obtain the result, but as a secondary preference want to withhold the result from as many other players as possible. The key insight is that at the end of protocol when every player has obtained a share of the result, no one is inclined to share his result, since it can only increase the other players chances to get the result, but not his own. They circumvented this problem with an elegant randomized protocol, that repeats the exchange until a final randomly chosen round and excludes all players that do not submit their share from subsequent rounds. This result was later improved in [1].

Since we use server-assisted SMC, we circumvented the protocol using the server. The server, as a service provider, has a different economic motivation and can act as a dealer in exchanging the results. Thereby our protocol achieves a fixed, constant number of rounds when all players are rational.

SMC when all players are rational for any mechanism to be implemented has been realized in [17]. They present a very clever protocol that can implement any mechanism regardless of the utility functions of the participants, i.e. they have no restriction as compared to [13]. This is a major break-through in DAMD and fully addresses the challenge of combining SMC and mechanism design. In fact, one can conclude that any game for which there is an ideal mechanism (or one

where truth-telling is dominant strategy) using a third party can now be implemented using this protocol. Unfortunately the protocol in [17] is currently not digitally implementable. It actually uses real-world building blocks, such as ballot-boxes and envelopes.

Our protocols do not preserve equilibria in all cases and therefore require restrictions on the utility function of the players. In particular, our protocols allow pre-game signaling in the collaborative coin flipping protocol, such that players could broadcast information about their input. This could modify the equilibria of a game.

A classification of functions that are computable using a trusted-third party in the presence of rational parties has been done for binary functions in [24]. Shoham and Tennenholtz show that (deterministic) binary function are computable by a trusted third party if the function is not reversible and not dominated. Although they do not prove extensions to integer valued functions, they conclude that simple arithmetic functions, e.g. sum or average are not computable by a trusted third party, while e.g. median is. This result has strong implications for all rational SMC protocols. Both our result and Halpern and Teague only apply to functions that are computable by trusted third parties. For the protocol by Izmalkov et al. it makes mechanism design trivial, since then only the function needs to be realized in the protocol.

3. Preliminaries

3.1. Yao’s Garbled Circuits

This section briefly describes the circuit encryption from [25]. This technique builds the basis for the protocol described here, as well as for many of its related work [2], [21].

Let \mathbb{C} be a binary circuit consisting of gates $\mathbb{G} = \{g_1, \dots, g_\alpha\}$ and wires $\mathbb{W} = \{w_1, \dots, w_\beta\}$. Let $\mathbb{I} = \{i_1, \dots, i_\gamma\} \subset \mathbb{W}$ be the input wires and $\mathbb{O} = \{o_1, \dots, o_\delta\} \subset \mathbb{W}$ be the output wires. The two input wires of each binary gate g_i are denoted as in_i^1 and in_i^2 and the output wire as out_i .

| | | |
|---|---|--|
| 0 | 0 | $E_{H(k_{in_1^1,0}, k_{in_2^2,0}, i)}(k_{out_i, r_{out_i}} \oplus op_{i, r_{in_1^1} \oplus 0, r_{in_2^2} \oplus 0})$ |
| 0 | 1 | $E_{H(k_{in_1^1,0}, k_{in_2^2,1}, i)}(k_{out_i, r_{out_i}} \oplus op_{i, r_{in_1^1} \oplus 0, r_{in_2^2} \oplus 1})$ |
| 1 | 0 | $E_{H(k_{in_1^1,1}, k_{in_2^2,0}, i)}(k_{out_i, r_{out_i}} \oplus op_{i, r_{in_1^1} \oplus 1, r_{in_2^2} \oplus 0})$ |
| 1 | 1 | $E_{H(k_{in_1^1,1}, k_{in_2^2,1}, i)}(k_{out_i, r_{out_i}} \oplus op_{i, r_{in_1^1} \oplus 1, r_{in_2^2} \oplus 1})$ |

Table 2. Encrypted truth table of gate g_i

For each wire w_i a random bit r_{w_i} is chosen and two random keys $k_{w_i,0}$ and $k_{w_i,1}$, such that the last bit of $k_{w_i,0}$ is 0 and the last bit of $k_{w_i,1}$ is 1. Each gate g_i can be represented as a truth table. Let $op_{i,a,b}$ be the result of the gate g_i 's operation on a and b . The garbled and encrypted truth table of gate g_i is displayed in Table 2.

A circuit where each gate has been encrypted this way can be executed, if the executor has one key per input wire. He can decrypt one entry in the truth table and will have one key of the output wire. Furthermore, if he has only one key per input wire, he will also only have one key for each other wire including the output wires and he will not be able to gain more information. For a proof of Yao's protocol and this technique see [20].

4. Protocol

In our protocol the server executes the circuit encrypted by the clients. The protocol proceeds as follows:

Setup: The clients C_i have agreed on a random number r unknown to the server S .

- 1) Using r (as a seed in a pseudo-random number generator) each client C_i chooses the random bits r_i and the keys $k_{i,0}$, $k_{i,1}$ and builds the encrypted circuit. Set r_o to 0 for $o \in \mathbb{O}$ (i.e. do not garble the result).
- 2) Let α be the number of gates $\mathbb{G} = \{g_1, \dots, g_\alpha\}$ in the circuit (wlog let $\alpha \bmod n = 0$). The order of the gates does not matter. Let the set $\mathbb{G}_i = \{g_{\frac{\alpha(i-1)}{n}+1}, \dots, g_{\frac{\alpha i}{n}}\}$ be C_i 's fraction of the circuit \mathbb{C} . Let $x_{i,j}$ be the bit of C_i 's input on wire j ($j \in \mathbb{I}_i \subset \mathbb{I}$). Each client C_i sends \mathbb{G}_i , $H(\mathbb{G} \setminus \mathbb{G}_i)$ and $k_{j,r_j \oplus x_{i,j}}$ for $j \in \mathbb{I}_i \subset \mathbb{I}$ to the server S .
- 3) The server S verifies the correctness of each hash $H(\mathbb{G} \setminus \mathbb{G}_i)$ given the received \mathbb{G}_i s. If they all match, the protocol continues. Otherwise the procedure to detect malicious clients from Section 4.0.1 is initiated. The server S executes the circuit \mathbb{C} and directly obtains the result f . He distributes f to each client C_i . If an entry of a

gate cannot be decrypted, the procedure to detect malicious clients from Section 4.0.1 is initiated as well.

The network complexity is $O(\frac{\alpha}{n})$ communication over linearly many ($O(n)$) links. For most practical problems interesting to current e-commerce providers, such as auctions (best-price, second-price, etc.) or benchmarking, the circuit size α is linear in the number of participants: $\alpha = O(n)$. Then the algorithm is "network efficient" with a constant communication cost over linearly many links.

4.0.1. Detection of Maliciously Modified Circuits.

Each client C_i sends a hash of the circuit excluding his fraction to the server S . The server can verify by recomputing the hashes that every client did indeed send the correct fraction of \mathbb{G} . If a client maliciously modifies his fraction (and all other clients do not collude with him), the hashes of all other clients will be incorrect.

To detect which client misbehaved and exclude that client from future computations, the server asks all clients to reveal \mathbb{G} . He can then exclude the malicious party (or parties) by comparing the submitted \mathbb{G}_i s to the majority. This procedure works as long as only a minority $t < \frac{n}{2}$ of clients is malicious.

If a decryption of a gate entry failed (because a client send a false key for his input), the server S asks all clients to reveal all $k_{i,0}$. He can then compare the inputs to the majority again and exclude false submitters.

A majority of malicious clients can force the server to exclude an honest minority using this procedure. Note, that at this point in the protocol no random bits r_i have been revealed and that privacy against $t = n-1$ clients is still maintained.

This procedure is also particularly important against rational attackers. Assume a rational attacker, that is interested in obtaining the result and withholding the result from other clients. The details of the utility function are given in Section 5.2. Such an attacker would modify the random bits r_o for all output wires w_o ($w_o \in \mathbb{O}$) in his fraction $w_o \in \mathbb{G}_i$. He could

obtain the correct result, since he knows r_o , but all other clients (and the server) would be oblivious to the correct result (or a fraction of it). Since, he now will be excluded if he attempts this attack, his rational behaviour changes to complying with the protocol.

Revealing \mathbb{G} in order to detect the malicious clients can be avoided. If each client C_i sends $n - 1$ hashes $H(\mathbb{G}_1), \dots, H(\mathbb{G}_{i-1}), H(\mathbb{G}_{i+1}), \dots, H(\mathbb{G}_n)$ to the server, the server can determine the malicious clients from the majority of the corresponding hashes for its fraction of \mathbb{G} . Nevertheless this protocol has communication complexity $O(\max(\frac{\alpha}{n}, n))$ over linearly many links which has a lower bound of $O(n)$ over linearly many links. Asymptotically this is not “network efficient” any longer. In practice due to the small size of the cryptographic hashes, this protocol can still be efficient.

Note, that $t = \lceil \frac{n}{2} - 1 \rceil$ malicious attackers may increase the complexity of the protocol to $O(\alpha n)$ if they collude in the worst case. A new circuit would be generated in all t rounds. Using the technique described in the paragraph above this can be reduced to $O(\alpha + n^2)$ where the circuit is kept even in the presence of malicious parties. Related work [5], [6], [16] considers only the case of keeping the circuit, but then also keeps its complexity. In case of a circuit regeneration the complexity would also increase by a factor of $O(n)$. The decision whether the circuit should be kept or regenerated when a malicious party is detected depends on the application. E.g. in an auction scenario the circuit should be kept, since a party should not be able to retract its bid by deviating from the protocol, while in a benchmarking scenario the circuit should be regenerated, because a malicious party may try to distort the result.

5. Analysis

5.1. Security

Security can be divided into security against semi-honest and malicious attackers. Semi-honest attackers follow the protocol as described, but keep a record of the interaction. They then try to infer as much information as possible about the other party’s input from this record. A malicious attacker is allowed to deviate in arbitrary ways from the protocol. Certain attacks, such as input substitution and early abort, cannot be prevented in the presence of malicious attackers. Both are models of computational security, as they are based on computational indistinguishability of ensembles. This is important, since our protocols rely on computational assumptions, such as the hardness

of the discrete logarithms as well. For an in-depth discussion of semi-honest and malicious security see [12].

The security of the protocol against semi-honest attackers follows from security of Yao’s protocol as proven in [20]. No information about the parties’ input other than the circuit and keys is revealed to no party other than the server. We can therefore state two theorems about the security of the server-assisted SMC protocol in the semi-honest model.

Theorem 1: Let f be a n -ary function. The server-assisted SMC protocol $n - 1$ -privately computes f in the semi-honest model if the server is honest.

Proof Sketch: All clients C_i receives only one message in the collaboration protocol, which is the result of f . They therefore cannot infer any more information, since the result is always revealed. \square

Theorem 1 states that no collusion of up to $n - 1$ clients can infer anything about any other party’s input.

Theorem 2: Let f be a n -ary function. The server-assisted SMC protocol 1-privately computes f in the semi-honest model.

Proof Sketch: For clients it is a corollary of Theorem 1. For the server the proof follows from the proof in [20]. \square

Theorem 2 states the server does not receive any information about the inputs as well (as long as he does not collude).

Furthermore the protocol is secure against malicious clients as well as described in Section 4.0.1.

Theorem 3: Let f be a n -ary function. The server-assisted SMC protocol securely computes f , if $t < \frac{n}{2}$ clients are malicious and the server is semi-honest.

Proof Sketch: If a minority $t < \frac{n}{2}$ of clients maliciously submits false input, the detection procedure of Section 4.0.1 excludes them from the protocol. The majority can complete the protocol. \square

Theorem 3 states that a minority of clients may fail arbitrarily. Malicious behaviour by the server can be prevented as well. The server’s only chance to maliciously (and undetectably) influence the protocol is when sending the result. If the server delivers a proof with the result, that shows that the result has been computed according to the circuit \mathbb{C} , the clients can verify the server’s honesty. Such a proof can consist of the server sending the keys k_o for the output wires ($o \in \mathbb{O}$) to the clients. Then the following theorem can be stated:

Theorem 4: Let f be a n -ary function. The server-assisted SMC protocol (augmented with an additional proof by the server) 1-securely computes f in the malicious model.

Proof Sketch: For the clients it is a corollary of Theorem 3. If the server could successfully forge the keys k_o , he could decrypt the circuit which contradicts the proof of [20]. \square

We show in the next section that protocol-compliant behaviour (as required by semi-honest security) is rational for the server. We therefore anticipate implementations of the protocol to operate with semi-honest servers. This is in contrast to the situation for clients, for which we have seen in Section 4.0.1 malicious behaviour is rational if the cryptographic hashes to verify the integrity of the circuit are omitted.

5.2. Rational Behaviour

Assuming rational behaviour of participants in the protocol has given rise to recent interesting results in the literature [11], [13], [17], [24]. We consider the case of rational behaviour of clients and server, i.e. everybody acting in his own best interest. This is separate from semi-honest and malicious security definitions. We assume that each client is interested in learning the correct result of the computation, and that as a second preference he is interested in withholding the result from other clients. This is the same setup as considered in [13], [24], but our protocol is distinguished by the presence of the server. The server acts as a service provider in the protocol and the economic transaction. We assume that the server is being paid for his services, and that it is in his best interest to deliver the correct result to the paying clients.

Before describing the utility of the players we review some notions from game theory. A *strategy* for client C_i or server S is a (possibly randomized) function of local information to actions. A *joint strategy* $\vec{\sigma} = (\sigma_1, \dots, \sigma_{n+1})$ is a tuple of strategies, one for each client C_i (σ_i) and one for the server S (σ_{n+1}). Let $U_i(\vec{\sigma})$ denote player i 's (server or client) expected utility, if $\vec{\sigma}$ is played.

Let $\vec{\sigma}_{-i}$ denote a tuple consisting of each players strategy in $\vec{\sigma}$ other than player i 's. A *Nash equilibrium* is a joint strategy, such that no player has any incentive to do anything different (given what the other players are doing). Formally, $\vec{\sigma}$ is a Nash equilibrium, if for all players i and strategies σ'_i : $U_i(\vec{\sigma}_{-i}, \sigma_i) \geq U_i(\vec{\sigma}_{-i}, \sigma'_i)$.

A game can have many Nash equilibria, some of which are quite unrealistic in practice. Game theory therefore came up with many improvements of the Nash equilibrium. We focus here on one widely used in DAMD [11]: A *weakly dominant strategy* is a strategy that, regardless of what any other players do, earns a player a payoff at least as high as any other strategy, and, that earns a strictly higher payoff for some other

players' strategies. Formally, we say that strategy σ_i is weakly dominant, if there exists a tuple of strategies of other players $\vec{\sigma}'_{-i}$, such that $U_i(\vec{\sigma}_{-i}, \sigma_i) > U_i(\vec{\sigma}'_{-i}, \sigma_i)$ and for all tuples of strategies of other players $\vec{\sigma}'_{-i}$ and all strategies τ we have $U_i(\vec{\sigma}'_{-i}, \sigma_i) \geq U_i(\vec{\sigma}'_{-i}, \tau)$.

A function f is *non-cooperatively computable* if there exists a weakly dominant strategy, such that clients C_i can compute f using a trusted third party. In more detail, f is (deterministically) non-cooperatively computable, if for any client C_i , every strategy σ_i and every input x_i of i , either there exists an outcome of f , such that σ_i does not compute that outcome, or else the outcome is oblivious to σ_i .

Let r and r' be a runs in the game tree. Let $info(r)$ be a tuple (s_1, \dots, s_n) where s_i is 1 if client C_i receives the correct result from the server, and is 0 otherwise. Let $info_i(r) = s_i$. The assumptions on the utility function of the clients are as in [13].

- C1. $u_i(r) = u_i(r')$ if $info(r) = info(r')$
- C2. If $info(r) = 1$ and $info(r') = 0$, then $u_i(r) > u_i(r')$
- C3. If $info_i(r) = info_i(r')$, $info_j(r) \leq info_j(r')$ for all $j \neq i$, and there is some j , such that $info_j(r) < info_j(r')$, then $u_i(r) > u_i(r')$.

Let $compute(r)$ be a tuple (s_1, \dots, s_n) where $s_i = 1$ if client C_i 's input is included in the computation, and is 0 otherwise. Let $\overline{compute}(r)$ denote the element-wise negation of $compute(r)$. The assumption on the utility function of the server is as follows:

- S1. $u_S(r) > u_S(r')$ if $\sum_{i=1}^n compute(r) \wedge info(r) > \sum_{i=1}^n compute(r') \wedge info(r')$
- S2. $u_S(r) < u_S(r')$ if $\sum_{i=1}^n \overline{compute}(r) \wedge info(r) > \sum_{i=1}^n \overline{compute}(r') \wedge info(r')$

We state the following theorem about the rational security of the server-assisted SMC protocol:

Theorem 5: If the clients' utilities satisfy C1-C3 and the server's utility satisfies S1-S2, then protocol compliant behaviour in the server-assisted SMC protocol for non-cooperatively computable functions f is a weakly dominant strategy for all players (server and clients).

Proof Sketch: Consider the alternative choices for a client, and assume that a client submits false input or no input (which is equal to deviating from the protocol in the malicious model), he then always receives a lower pay-off (no matter what the server does), since $info_i(r) = 0$. This follows from the definitions of non-cooperative computation in [24]. For the server delivering the computed result to all not excluded (as in Section 4.0.1) clients only has a higher pay-off than all other strategies if all not excluded clients submit the correct input. No strategy can have a higher utility,

since the maximum number of not excluded clients and the minimum number of excluded clients learns the result. If the server does not exclude malicious clients, he reduces his pay-off, since the number of clients that will learn the correct result is reduced. Consequently excluding malicious clients maximizes the number of clients that will receive the correct result. \square

Note, that if assumption S2 is omitted, then it is rational for the server to deliver the result to all clients (even excluded ones), yet still exclude malicious clients from the computation. This business case could be extended to interested third parties who purchase just the result. Nevertheless we decided that it is critical for the server to punish misbehaving clients to reduce the computational cost, i.e. reduce the number of necessary reruns of the protocol.

5.3. Anonymity

Anonymity can be very important for the clients, server or application. Clients may not want to reveal their participation in the protocol. For the server not having to reveal its customers to every client is a clear economic benefit. Certain applications may only be possible with anonymization, e.g. multi-group benchmarking [19].

Anonymity can be broken by any static identifier, e.g. IP addresses or public keys. A static identifier is a piece of information specific to an entity that does not change between multiple runs of the protocol. Even pseudonymous identifiers such as public keys reveal the composition of the clients by comparing multiple runs.

Our server-assisted SMC protocol efficiently allows for a certain type of anonymity. The clients in the protocol may remain anonymous amongst each other, but the server will be known to every client. This achieves all three privacy goals: clients, server, and application.

- The clients do not reveal their participation in the protocol (except to the server).
- The server does not have to reveal his customers to any client.
- The participants in the application (clients) remain anonymous.

Furthermore, it fits our economic motivation where the server charges the clients for the service. He knows the identity of each client and can charge them, but does not have to reveal them.

We can therefore state the following theorem

Theorem 6: The clients of the server-assisted SMC protocol stay anonymous amongst each other.

Proof Sketch: The communication in all protocols is only between a client and the server, i.e. no clients exchange messages directly. Therefore the communication does not break anonymity among the clients (e.g. by IP addresses).

The clients only identifying information in the protocol is the identifier i chosen by the server in the collaborative anonymous identification protocol. This identifier is known only to the client and the server and is not shared explicitly or implicitly. All key information is shared among all clients. Therefore the protocols themselves do not break anonymity among the clients.

In summary, neither the primitives used nor the protocols break anonymity. Therefore the clients remain anonymous during the server-assisted SMC protocol. \square

6. Conclusions

This paper presented a novel SMC protocol that uses a central server to coordinate the protocol. The central server allows for a more practical and efficient network coordination of the protocol. It is the first SMC protocol that is optimistic, i.e. has a better communication complexity if all parties behave semi-honest. This is in contrast to traditional SMC protocols where the worst-case complexity equals the best-case complexity.

The protocol is secure against rational players that want to obtain the correct result and as a second preference withhold it from as many other players as possible. Furthermore we assume a service provider model where the server wants to deliver the correct result to as many cooperating clients as possible, because he is getting paid for it.

We therefore believe that our protocol is an important step towards solving the open problem of “network efficient” private mechanism as posed by the DAMD literature [11]. In particular our server-assisted SMC protocol is the first “network efficient” protocol. It has only a constant communication complexity over a linear number of links.

Furthermore, despite previous impossibility results [13], we achieve a constant, fixed number of rounds. This is made possible by the introduction of the server that, as a service provider, has different incentives. The server acts a service provider to facilitate distribution of the results.

Another important additional feature by our protocol is anonymity of the clients amongst each other. This is a rarely investigated feature, but imperative for the adoption of the service provider model. The service

provider does not have to reveal the identity of its customers to each other.

Our protocol is also the first to adopt a single server SMC protocol. We believe that multiple server SMC protocols are not fit for the adoption in business practice, since they require a different business model where benefits are shared. In our protocol the service provider is able to offer the service by himself.

7. Acknowledgements

The developments presented in this paper were partly funded by the European Commission through the ICT program under Framework 7 grant FP7-213531 to the *SecureSCM* project.

References

- [1] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation. *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, 2006.
- [2] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols. *Proceedings of the 22nd ACM Symposium on Theory of Computing*, 1990.
- [3] 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.
- [4] C. Cachin. Efficient Private Bidding and Auctions with an Oblivious Third Party. *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.
- [5] R. Cramer, I. Damgard, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. *Proceedings of Eurocrypt*, 2001.
- [6] I. Damgard, and J. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. *Proceedings of Crypto*, 2007.
- [7] G. Di Crescenzo. Private Selective Payment Protocols. *Proceedings of Financial Cryptography*, 2000.
- [8] G. Di Crescenzo. Privacy for the Stock Market. *Proceedings of Financial Cryptography*, 2001.
- [9] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the Cost of Multicast Transmissions. *Journal of Computer and System Sciences* 63, 2001.
- [10] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure Computation of Surveys. *Proceedings of the EU Workshop on Secure Multiparty Protocols*, 2004.
- [11] J. Feigenbaum, and S. Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. *Bulletin of the EATCS* 79, 2003.
- [12] O. Goldreich. Secure Multi-party Computation. Available at www.wisdom.weizmann.ac.il/~oded/pp.html, 2002.
- [13] J. Halpern, and V. Teague. Rational Secret Sharing and Multiparty Computation: Extended Abstract. *Proceedings of the 36th ACM Symposium on Theory of Computing*, 2004.
- [14] M. Harkavy, D. Tygar, and H. Kikuchi. Electronic Auctions with Private Bids. *Proceedings of 3rd USENIX Workshop on Electronic Commerce*, 1998.
- [15] M. Hirt, U. Maurer, and B. Przydatek. Efficient Secure Multi-Party Computation (Extended Abstract). *Proceedings of Asiacrypt*, 2000.
- [16] M. Hirt, and J. Nielsen. Robust Multiparty Computation with Linear Communication Complexity. *Proceedings of Crypto*, 2006.
- [17] S. Izmalkov, S. Micali, and M. Lepinski. Rational Secure Computation and Ideal Mechanism Design. *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, 2005.
- [18] A. Juels, and M. Szydlo. A two-server, sealed-bid auction protocol. *Proceedings of the 6th Conference on Financial Cryptography*, 2002.
- [19] F. Kerschbaum. Building a Privacy-Preserving Benchmarking Enterprise System. *Proceedings of EDOC Conference*, 2007.
- [20] Y. Lindell, and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Electronic Colloquium on Computational Complexity* 11, 2004.
- [21] M. Naor, B. Pinkas, and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999.
- [22] S. Pohlig, and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory* 24, 1978.
- [23] B. Schneier. Applied Cryptography (2nd edition). *John Wiley & Sons*, 1996.
- [24] Y. Shoham, and M. Tennenholtz. Non-Cooperative Computation: boolean functions with correctness and exclusivity. *Theoretical Computer Science* 343(1-2), 2005.
- [25] A. Yao. Protocols for Secure Computations. *Proceedings of the annual IEEE Symposium on Foundations of Computer Science* 23, 1982.