A Single Initialization Server for Multi-Party Cryptography

Hugue Blier and Alain Tapp

Département d'informatique et de recherche opérationnelle
Université de Montréal, C.P. 6128, Succ. Centre-Ville
Montréal (QC), H3C 3J7 Canada

Abstract. We present information-theoretically secure bit commitment, zero-knowledge and multi-party computation based on the assistance of an initialization server. In the initialization phase, the players interact with the server to gather resources that are later used to perform useful protocols. This initialization phase does not depend on the input of the protocol it will later enable. Once the initialization is complete, the server's assistance is no longer required. This paper improves on previous work as there is only one server and it does not need to be trusted. If the server is honest, the protocols are secure against any coalition of dishonest players. If all players are honest, then there is an exponentially small probability that both the initialization phase succeeds and that later the protocol fails. That is, the server cannot create a situation in the initialization phase that would lead honest players to accuse each other. The protocols are built in a modular fashion and achieve linear complexity for the players in terms of the security parameter, number of players and the size of the circuit.

keywords: two-party computation, multi-party computation, cryptography, zero-knowledge, initialization server.

1 Introduction

Two-party computation is a common scenario: Alice and Bob want to compute a function based on their inputs such that they get the correct output but also without revealing their respective input to the other participant. This situation can obviously be generalized to more than two participant. Multi-party computation was first introduced by [24, 25, 18]. It has been shown that without any computational assumptions, secure multi-party computation is possible if and only if a majority of participants are honest, in the presence of a broadcast channel [22]. If no broadcast channel is available, this proportion must be strictly more than 2/3 [3, 6]. Multi-party computation security can also be based on other assumptions: noisy channels [14, 11, 15], or using directly some primitives such as oblivious transfer (OT) [19, 13], trapdoor one-way permutations [18] or bounded memory [5, 4].

Beaver [1, 2] introduced in 1997 a model where a server is involved in the computation. This third party is said to be semi-trusted. It is trusted in the sense that it doesn't collude with any participant and it follows the protocol correctly. But if the players are honest, a dishonest server cannot learn anything about the input and output of the protocol it enables. In some contexts, the qualifier *honest but curious* is also used. Beaver uses this server to distribute *commodities* to users prior to the calculation. Under these assumptions, he realizes a protocol for OT on which secure multi-party computation can be based. In the same model, Rivest [23] has also shown simple algorithms for bit commitment (BC) and OT. Many specific problems having practical applications have been solved using such a third party [10, 16, 17]. This model is very appealing since it is close to the Internet setting in which a server provides services. Since this server does not have to be fully trusted, it has practical applications.

Yet trust is an issue and this is the problem we address in this paper. In [1, 23], since the server has to follow the protocol, it is an issue to choose a server trusted by both parties. The way they addressed this problem was by using more than one server. The drawback is that the protocol is less practical. Here, our protocols deal with a dishonest server, as long as it doesn't collude with other participants. That is, the server could make the initialization phase fail, but will not be able to make honest players accuse each other of cheating. Once the initialization phase succeeds, the security of the primitives and protocols performed in the computation phase is unconditional.

Our protocols have the following properties. If the server does not collude with any player, the initialization phase enables protocols that achieve information theoretical privacy and correctness. That is, the protocols are resilient to both cheating players and a dishonest server at the same time under the no-collusion assumption. If all players are honest, the server will gain no information about the protocol realized after the initialization phase. Furthermore, whatever the server behaviour, the probability that both the initialization phase succeeds and the later protocol aborts is exponentially small. The last criteria is unusual in conventional multi-party computation but is reasonable in presence of two different types of actors.

An appealing aspect of our protocols is their simplicity and efficiency. As first said, the situation of multi-party computation has been studied for a long time and is well understood. An attentive reader will recognize flavours of known techniques. For example, our protocols are based on so-called commitment chips similar to two precomputed oblivious transfers and our bit commitments are constructed so that the Rudich technique [20, 13, 12] can be used and some noise can be tolerated.

It is worthwhile to mention that a simple but inefficient solution can easily be obtained from known techniques. For example the server, during the initialization phase, could distribute two random strings of n bits to Alice and Bob such that the Hamming distance between these two strings is ϵn. This could be probabilistically verified with some accuracy. Afterwards, Alice and Bob could use these two strings as one-time pads to communicate. This would result in a binary symmetric channel with error ϵ which is known to be sufficient for multi-party computation [14, 13] since it enables the participants to realize OT. Another way would be to adapt ideas from [9] and [8]. It is not too hard to obtain similar results based these article for the two-party case, but for the multi-party case, the obtained protocol would be significantly less efficient than the one we present. Note also that [21] propose a elegant solution where the initialization server is fully trusted; in our protocol, the server does not have to be trusted and the solution we propose is also more efficient.

In the following sections, we present protocols for commitment (Sect. 2), committed circuit evaluation and zero-knowledge (Sect. 3) and multi-party secure computation (Sect. 4). Even though our protocols are intricate, the proofs are relatively straightforward and are not particularly enlightening. Because of lack of space, they will not be presented in this extended abstract.

2 Bit Commitment

BC is a cryptographic procedure composed of two phases. In the commitment phase, Alice commits to a bit value with Bob and in the opening phase, she reveals that bit. We say that the commitment is binding if, after the commitment phase, Alice can only open one unique value. We say that the commitment is concealing if, after the commitment phase, Bob has no information about the committed bit. Note that the opening phase is optional.

To accomplish BC (as well as all the following protocols), we rely on commitment chips (CCs). Our protocol begins by an initialization phase where the server creates enough CCs and gives them to the players. A CC i is a weak commitment to the value $v_i = x_1^i \oplus x_2^i \oplus x_3^i \oplus x_4^i$, the parity of four bits that the server privately transmits to Alice. Of these four bits, the server only transmits to Bob one of the first two and one of the last two. We will always suppose that communication between the players and the server is done in a private way. CCs can be seen as a combination of 2 $\binom{1}{2}$-OTs are constructed is such a way that the Rudich technique can be used. It is crucial that Alice doesn't know which bits Bob knows. The CCs created in the initialization phase are the resources shared by Alice and Bob to construct BCs and all other protocols.

In the protocols, we denote Alice by $\mathcal{A}$, Bob by $\mathcal{B}$ and the server by $\mathcal{S}$. Note that except if otherwise stated, the CCs and BCs are from Alice to Bob.

Protocol 1 CC Commit

Input: an index $i \in I$
Result: the CC indexed by i is created

```
$\mathcal{S}$ chooses $x_1^i, x_2^i, x_3^i, x_4^i \in_R \{0,1\}$ and sends them to $\mathcal{A}$
$\mathcal{S}$ chooses $\ell^i \in_R \{1,2\}$ and $r^i \in_R \{3,4\}$ and sends to $\mathcal{B}$ $(\ell^i, r^i, x_{\ell^i}, x_{r^i})$
```

To verify the honesty of the server (i.e. that the bits of Alice and Bob correspond), half of the CCs given by the server will be opened. In all our protocols, we say that a bit is *inconsistent* whenever Alice and Bob disagree on its value.

Protocol 2 CC Unveil

Input: an index $i \in I$
Result: the CC indexed by i is unveiled

```
$\mathcal{A}$ sends $x_1^i, x_2^i, x_3^i, x_4^i$ to $\mathcal{B}$
$\mathcal{B}$ outputs FAIL if $x_{\ell^i}^i$ or $x_{r^i}^i$ are inconsistent
```

From the protocol **CC Preprocessing** we can already see the role of the two bits given to Bob: if Alice wants to change the value of one CC, she must change the value of at least one its four x_is. Since she is not aware of which bits Bob knows, she will change a bit Bob knows with probability one-half, and get caught. Note also that since Bob knows only two bits of each CC, he has no information about the parity of the four bits.

Protocol 3 CC Preprocessing

Result: I an index set of CCs

```
Let $I$ be a set of indices
$\forall i \in I$, Call CC Commit($i$)
$\mathcal{B}$ chooses $O \subset_R I$ such that $|O| = \frac{|I|}{2}$ and sends its description to $\mathcal{A}$
$\forall i \in O$, Call CC Unveil($i$) and Bob outputs ABORT if the output is FAIL
$\mathcal{A}$ and $\mathcal{B}$ set $I$ to $I \setminus O$
```

Since we would like Alice to only have an exponentially small probability of successfully cheating when committing, we define $s = 2k + 1$ (a odd security parameter), and a BC to the value b will be a group of $3s$ CCs to the value b. The choice of $3s$ instead of s is useful in the following section. Note that once the initialization phase is complete, the players do not need the server to realize BC.

After the initialization phase, a set of indices I corresponding to CCs is shared between Alice and Bob. To construct a BC (as well as other protocols), CCs are consumed and removed from this set.

Protocol 4 BC Commit

Input: $b \in \{0, 1\}$ and I an index set of CCs
Result: a BC B to the value b (I is updated)

```
$\mathcal{A}$ chooses $B \subset_R I$ such that $|B| = 3s$ and such that $\forall i \in B, v_i = b$
$\mathcal{A}$ sends a description of $B$ to $\mathcal{B}$
$\mathcal{A}$ and $\mathcal{B}$ set $I$ to $I \setminus B$
```

To open the BC C, Alice only needs to reveal every bit of every CC. The condition for Bob to accept the opening is that no more than $\frac{1}{10}$ of the CCs aren't consistent with the bits he knows. Why? Because the server is not trusted. The verification done in the initialization phase assures the players that there is little inconsistency, but not that there is none.

If Alice is dishonest, she can choose to construct a BC in an *undefined* way by choosing CCs with two different values. In order to ensure that the BC value is always well-defined, we say that the value of a BC is the value of the majority of the values of the CCs of which it is made. This is why we choose s to be odd.

Protocol 5 BC Unveil

Input: a BC B
Result: B is opened

```
$\mathcal{A}$ sends $b$ to $\mathcal{B}$
$\mathcal{B}$ sets $e$ to 0
$\forall i \in B$
    Call CC Unveil($i$)
    if CC Unveil($i$) outputs FAIL or does not have value $b$, then set $e$ to $e + 1$
If $e \geq \frac{|C|}{10}$, $\mathcal{B}$ outputs ABORT
```

Usually, in the analysis of a two-party protocol, we consider what happens when one of the participants is honest and the other is dishonest. Here, we also have to consider the fact that the server can be dishonest.

Lemma 1. *(**BC Commit**: concealing) As long as Bob and the server do not collude, after **BC Commit**(b), Bob has no information on b.*

Lemma 2. *(BC: binding) As long as Alice and the server do not collude, after **BC Commit**(B), **BC Unveil**($B = \bar{b}$) has a chance exponentially small in s to succeed.*

Since we want to consider a cheating server, we also want to be sure that, if the server is dishonest, none of the honest players can be falsely incriminated.

Lemma 3. *(BC: robust) Given that Alice and Bob are honest, there is an exponentially small probability that both the preprocessing succeeds and that one of the later **BC Unveil** aborts.*

So, if Alice and Bob are honest and the server is dishonest, it cannot make the initialization phase succeed in such a way that later, a commitment phase or an opening phase will fail. Once the initialization phase has been done, the only way for the protocol to abort is if Alice or Bob misbehave. Thus, there is no way the server can cheat in a way that Alice or Bob will be accused wrongly (except with exponentially small probability).

A very useful characteristic of our BC protocol is the possibility for Alice to choose m BCs and prove to Bob that the parity of their committed values is p without revealing any other information. The parity of m CCs, each chosen from a different BC, must also be p. The protocol **CC Parity** verifies this fact, but has probability 1/2 of failure in case Alice tries to cheat. **BC Parity** simply calls **CC Parity** s times to amplify this probability. This is the well-known technique introduced by Rudich. At the end of the **BC Parity** protocol, Bob will be convinced of the parity and all the BCs will remain valid, but s CCs contained in each BC will have been consumed.

Protocol 6 CC Parity

Input: a list of CC indices $i_1, i_2, \ldots, i_m$ and a parity p
Result: $\mathcal{B}$ learns if $p = v_{i_1} \oplus v_{i_2} \oplus \cdots \oplus v_{i_m}$

```
$\mathcal{A}$ sends $p$ to $\mathcal{B}$
$\mathcal{A}$ computes $q = \bigoplus_{j=1}^{m} x_1^{i_j} \oplus x_2^{i_j}$ and sends it to $\mathcal{B}$
$\mathcal{B}$ sends $r \in_R \{0,2\}$ to $\mathcal{A}$
For $j$ from 1 to $m$, $\mathcal{A}$ sends to $\mathcal{B}$ $x_{1+r}^{i_j}$ and $x_{2+r}^{i_j}$
$\mathcal{B}$ checks that these values are consistent and:
If $r = 0$, $\mathcal{B}$ checks that $q = \bigoplus_{j=1}^{m} x_1^{i_j} \oplus x_2^{i_j}$
If $r = 2$, $\mathcal{B}$ checks that $p \oplus q = \bigoplus_{j=1}^{m} x_3^{i_j} \oplus x_4^{i_j}$
If an error is detected, the output of the protocol is FAIL
```

Note that FAIL is an acceptable outcome for a sub-protocol but that when a sub-protocol outputs ABORT, it implies that the calling protocol also outputs ABORT and so on. Note that each BC is composed of $3s$ CCs. This implies that three operations (**BC Parity**, **BC Unveil**) can be performed on a single BC before it becomes useless. This can be used to perform FAN-OUT and NOT gates in the obvious way.

Protocol 7 BC Parity

```
Input: a list of BCs $B_1, B_2, \dots, B_m$ and a bit $p$
Result: $\mathcal{B}$ learns if $p = b_1 \oplus b_2 \oplus \cdots \oplus b_m$

$\mathcal{A}$ computes $p = b_1 \oplus b_2 \oplus \cdots \oplus b_m$ and sends it to $\mathcal{B}$
$\mathcal{B}$ sets $e$ to 0
Repeat $s$ times
    For $j$ from 1 to $m$
        $\mathcal{B}$ chooses $i_j \in_R B_j$ and send it to $\mathcal{A}$
        $\mathcal{A}$ and $\mathcal{B}$ sets $B_j$ to $B_j \smallsetminus i_j$
    $\mathcal{A}$ and $\mathcal{B}$ call CC Parity$(i_1, i_2, \dots, i_m, p)$
    If the protocol outputs FAIL, set $e$ to $e + 1$
If $e > \frac{s}{10}$ then ABORT
```

Lemma 4. *(**BC Parity**: robust) Given that Alice and Bob are honest, the probability that both the initialization phase succeeds and that **BC Parity** outputs ABORT is exponentially small in s.*

Lemma 5. *(**BC Parity**: zero-knowledge) If Alice and the server are honest, after **BC Parity**$(B_1, B_2, \dots, B_m)$, Bob cannot learn any information except $p = b_1 \oplus b_2 \oplus \cdots \oplus b_m$.*

Lemma 6. *(**BC Parity**: sound) Given that Alice and the server do not collude and that $p = b_1 \oplus b_2 \oplus \cdots \oplus b_m$, the probability that Parity BC$(B_1, B_2, \dots, B_m, \overline{p})$ succeeds is exponentially small in s.*

3 Oblivious Circuit Evaluation and Zero-Knowledge

In this section, we discuss techniques called oblivious circuit evaluation (OCE) to compute functions on committed bits, producing committed bits with the right relationship without revealing any information whatsoever (i.e. in a zero-knowledge way). To do this, it is sufficient to be able to do a NOT gate and an AND gate on committed bits.

The NOT could be achieved using the **BC Parity** protocol from the previous section. One just has to observe that $x = y \Leftrightarrow x \oplus y = 0$. The implementation of the AND gate requires some preprocessing. This preprocessing step creates AND-Commitment-Chips (ACCs). An ACC is a triplet of commitment chips (V_1, V_2, V_3) such that $v_1 \wedge v_2 = v_3$. As you can se, we use uppercase letters for objects and lowercase letter for bit values. As usual, the preprocessing of the ACC is independent of the circuit to be evaluated later. Each AND gate requires the use of $10s$ ACCs. It is straightforward to see that with the tools to evaluate an arbitrary known circuit on committed values, it is easy to prove any statement in NP in a zero-knowledge way.

Suppose that Alice has commitments B_1 and B_2 and wants to commit herself to the AND of the two values. First, she creates ACCs to random values. Half of them are opened to ensure consistency. She then chooses a subset of the remaining ACCs with the appropriate value and then uses them to construct B_3 such that $b_1 \wedge b_2 = b_3$. In the protocol, the union of the third component of every ACC forms a BC to the desired value. For this to be secure, we have on one hand to manage potential errors but on the other hand to prevent Alice from using these to create a tweaked gate. Alice has therefore to group ACCs each time an AND gate is to be computed. The final protocol, **OCE** of a function F, can be realized using **BC Parity** to perform NOT and FAN-OUT gates and **OCE AND** for AND gates.

Protocol 8 OCE AND

Input: BCs B_1,B_2 and B_3

Result: $\mathcal{B}$ is convinced that $b_3 = b_1 \wedge b_2$

```
$\mathcal{A}$ chooses at random a set $T$ of triplets of indices of CCs such that
    $\forall [i_1, i_2, i_3] \in T, v_{i_3} = v_{i_1} \wedge v_{i_2}$
    $|T| = 10s$
    and sends it to $\mathcal{B}$
$\mathcal{B}$ sets $e$ to 0
$\mathcal{B}$ creates $O \subset_R T$ such that $|O| = 5s$ and sends it to $\mathcal{A}$
$\forall [i_1, i_2, i_3] \in O$
    $\mathcal{A}$ opens $i_1, i_2$ and $i_3$ using **CC Unveil**
    if any of the openings output FAIL or
    if $v_{i_3} \neq v_{i_1} \wedge v_{i_2}$ $\mathcal{B}$ sets $e$ to $e + 1$
If $e > \frac{5s}{10}$ then $\mathcal{B}$ outputs ABORT
$\mathcal{B}$ sets $e$ to 0
$\mathcal{A}$ creates $S \subset T$ such that $|S| = s$ and $\forall [i_1, i_2, i_3] \in S$,$v_{i_1} = b_1$ and $v_{i_2} = b_2$
$\mathcal{A}$ sends a description of $S$ to $\mathcal{B}$
$\mathcal{A}$ and $\mathcal{B}$ set $T$ to $T \setminus S$
$\forall [i_1, i_2, i_3] \in S$
    $\mathcal{B}$ chooses $j_1 \in_R B_1$, $j_2 \in_R B_2$ and $j_3 \in_R B_3$
    $\mathcal{A}$ proves that $v_{i_1} = v_{j_1}$,$v_{i_2} = v_{j_2}$ and $v_{i_3} = v_{j_3}$ using **CC Parity**
    If this FAILS then set $e$ to $e + 1$
    $\mathcal{A}$ and $\mathcal{B}$ set $B_1$ to $B_1 \setminus j_1$, $B_2$ to $B_2 \setminus j_2$ and $B_3$ to $B_3 \setminus j_3$
If $e > \frac{s}{10}$ then $\mathcal{B}$ outputs ABORT
```

Lemma 7. *(**OCE**: robust) Given that Alice and Bob are honest, the probability that both the initialization phase succeeds and that the **OCE** protocol aborts is exponentially small in s.*

Lemma 8. *(**OCE**: sound) If Bob and the server are honest, the value of the commitments at the end of **OCE** protocol are correct except with exponentially small probability in s.*

Lemma 9. *(**OCE**: zero-knowledge) If Alice and the server are honest, Bob learns absolutely nothing while performing **OCE** with Alice.*

4 Secure Multi-Party Computation

In this section, we deal with protocols that can involve more that two participants. Secure multi-party computation (MP) is a n player task where all players learn the output of a circuit evaluated on private inputs coming from each of them. This is a very general, useful and well-studied task. In this section, we present a protocol to implement information theoretically secure MP. As in our previous protocols, an initialization phase is required, but this phase does not depend on the function that is computed later. As usual in multi-party computation, all the input bits and the intermediate values are shared using secret sharing among all players. Since we have no bound on the number of dishonest players, the secret sharing scheme we choose is simply parity. In the first step, each player commits to his input. Then the circuit is evaluated gate by gate (AND gates and NOT gates), and the output is a committed secret shared value between all players. Finally, the answer bits are opened. Known techniques [7] could be used to reveal the result of the function gradually in order not to give an unfair advantage to any of the players. This process is called multi-party computation (MP).

We implement the inputs and all intermediate values by a distributed BC (DBC). A DBC B with value b is a set of n^2 BCs $B[i,j]$ (player i is committed to player j) with value $b[i,j]$ such that for all i, we have that for all j and j', $b[i,j] = b[i,j']$ and therefore we choose to denote that value by $b[i]$. In addition, $b = \bigoplus_{i=1}^{n} b[i]$. Of course, this is in the honest cases. Inconsistent DBCs will fail when used in an AND gate.

To begin the computation of the known circuit, each player i has to construct, with the help of the other players, a DBC to every b of his input bits. This is done by having player i commit to every other player to b ($\forall j, B[i,j] = b$) and all other players commit to 0 ($\forall k \neq i, j, B[k,j] = 0$). The consistency of the commitment of player i is not verified, but the fact that all other commitments are equal to zero is.

Protocol 9 MP bit initialization

Input: a bit d from player i
Result: a DBC B with value b.

```
$\forall j \neq i$
    Player $i$ commits to $b$ to Player $j$ using **BC Commit**
    $\forall k \neq j$
        Player $j$ commit to bit 0 to player $k$ using **BC Commit**
        Player $k$ chooses $s$ CCs in that commitment
        Player $j$ opens these CCs using **CC Unveil**
        If more than $\frac{s}{10}$ are inconsistent or do not have value 0, Player $k$ outputs ABORT
```

There is no way after **MP bit initialization** to be sure that player i is committed to the same value with all the players. If this is not the case, the first AND gate directly or indirectly involving this bit will fail, except with exponentially small probability.

Protocol 10 DACC

Result: a set D of DACC is created

```
Let $D$ be a set of indices
For all $k \in D$
    $\mathcal{S}$ chooses uniformly at random $v[i], v'[i]$ and $v''[i]$ such that
      $\bigoplus_{i=1}^{n} v''[i] = \bigoplus_{i=1}^{n} v[i] \wedge \bigoplus_{i=1}^{n} v'[i]$
    $\forall i, j \neq i$, $\mathcal{S}$ creates CCs from $i$ to $j$ to the value $v[i], v'[i]$ and $v''[i]$ using **CC Commit**,
    $v[i], v'[i]$ and $v''[i]$ are associated with the index $k$
$\forall j$, player $j$ chooses $1/2n$ indices in $D$ and asks the other players to unveil all these CCs
    If any of the unveiling FAILS or if $\bigoplus_{i=1}^{n} v''[i] \neq \bigoplus_{i=1}^{n} v[i] \wedge \bigoplus_{i=1}^{n} v'[i]$ then ABORT
All players remove the opened value from $D$
```

In the preprocessing stage, the server creates random triplets of vectors of CCs, called Distributed AND CCs (DACCs). These triplets are such that the AND of the first two values equals the third one.

Note that there is an exponential number of different possible distinct DACCs. Therefore, the server cannot just choose groups of identical ones to form a Distributed AND BC (DABC). Actually the same kind of problem will also happen in the choice of DABC. In order to have an efficient algorithm, we have to use a trick, both in **DABC** and **MP AND**.

Once the DACCs are verified, they will be grouped by the server to create a DABC. The server does not group them such that the vectors are identical but only such that the parity of the vectors is identical. Note that by flipping an even number of bits, the parity of a vector does not change. Thus, to make an identical set, the participants have to modify the vector (in an even number of places) such that at the end they are identical. All this is accomplished without interaction with the server and without revealing information on the committed bit of the resulting triplet.

Protocol 11 DABC

Input: D an index set of DACCs

Result: E a set of DABCs

```
The server chooses $C \subset_R D$ such that $|C| = s$ and
    $\exists \alpha, \beta, \forall k \in C,\ \alpha = \bigoplus_{i=1}^{n} v_k[i]$ and $\beta = \bigoplus_{i=1}^{n} v'_k[i]$
Let $k' \in C$ be a specific index
The server broadcasts the index forming a description of $C$ and the value $k'$
$\forall k \in C \setminus k'$, $\forall i$
    Player $i$ broadcasts $t[i] = v_k[i] \oplus v_{k'}[i]$ and $t'[i] = v'_k[i] \oplus v'_{k'}[i]$
    For every 1 broadcasted,
        each player flips the first bit of the four-tuple constituting the CC
If $\bigoplus_{i=1}^{n} t[i] \neq 0$ or $\bigoplus_{i=1}^{n} t'[i] \neq 0$ then ABORT
$D$ is set to $D \setminus C$ and $C$ is added to $E$
All the previous steps are repeated until there is are enough elements in $D$
```

Protocol 12 MP NOT

Input: a DBC B and a set E of DABCs

Result: a DBC B' such that $b \neq b'$

```
Let $\beta$ be the value that player 1 is committed to in $B$
$\forall i \neq 1$
    Player 1 commits with $B'[1,i]$ to value $\overline{\beta}$ using **BC Commit**
    Player 1 proves to player $i$ that $b'[1,i] \neq b[1,i]$ using **BC Parity**
    $\forall j$ Player $i$ and $j$ rename $B[i,j]$ to $B'[i,j]$
```

In order to process a NOT gate, one player (we arbitrarily choose player 1 here) commits to the opposite bit with every player. These new commitments replace his commitment in the DBC vector B.

In order to compute the AND, we will use the DABC. The idea is similar to the computation of an AND gate in the preceding section: choose a DABC such that the first two vectors are identical to those in the input of the gate and consider the third one as the output. Once again, an exponential number of DABCs would be needed to achieve a perfect matching. Fortunately, this is not required, we only need the number of differences to be even, which means the values are equal. This simple idea is very important in making the protocols efficient.

Protocol 13 MP AND

Input: two DBCs B and B', E a DABC
Result: a DBC A'' such that $b \wedge b' = a''$

```
Repeat
    Player 1 chooses $C \in_R E$ ($C = (A, A', A'')$) and announces his choice
    $\forall i$
        Player $i$ broadcasts $p[i] = b[i] \oplus a[i]$
        Player $i$ broadcasts $p'[i] = b'[i] \oplus a'[i]$
Until $\bigoplus_i p[i] = 0$ and $\bigoplus_i p'[i] = 0$
$\forall i$
    $\forall j \neq i$
        Player $i$ proves to player $j$ that $p[i] = b[i] \oplus a[i]$ using **BC Parity**
        Player $i$ proves to player $j$ that $p'[i] = b'[i] \oplus a'[i]$ using **BC Parity**
The $A''$ is the resulting DBC
```

As mentioned, there is no way during the initialization phase to be sure the one player is committed to the same value with every other player. This problem is solved because of the properties of the **MP AND** protocol. In that protocol, every player must broadcast the parity of his values in B (and B') and his value in A (and A'). If he had committed to an opposite value in the initialization phase, he must now cheat the parity protocols. We must assume that every bit is involved directly or indirectly in at least one AND gate (possibly a dummy one).

Combining the protocols presented in this section, we obtain a protocol **MP**. FAN-OUT is done in the obvious way, the NOT and AND gates use protocols **MP NOT** and **MP AND** and initialization of input bits using protocol **MP bit initialization**. The **MP** protocol we obtain has the following nice properties.

Lemma 10. *(**MP**: correct) If the server and all players are honest, the function is computed correctly with probability 1, in expected polynomial time.*

Lemma 11. *(**MP**: robust) Whatever the server does, if all players are honest, the probability that the preprocessing succeeds and MP fails is exponentially small in s.*

The previous lemma implies that if all players are honest it is almost impossible for the server to act in such a way that players will be led to believe that the protocols failed because of a dishonest player. Conversely, if a protocol fails after the initialization phase, this means that the most likely explanation is that a player cheated.

Lemma 12. *(**MP**: zero-knowledge) In MP, any group of dishonest players cannot learn anything else than the outcome of the function provided they do not collude with the server.*

5 Conclusion and Future Work

We have presented protocols for a server to provide resources to players so that they can perform at a later time protocols that implement bit commitment, zero-knowledge and secure multi-party computation. Our protocols and initialization phase are efficient, quite simple and their security is easy to verify. Table 1 summarizes the complexity, both in terms of communication and computation. The complexity of the protocols take into account the creation of necessary resources in the initialization phase. The complexity for the server in each of these protocols is obtained by multiplying by the number of players. We use n for the number of players, m for the size of the function and s for the security parameter.

Table 1. Protocol Complexity

Protocol	Expected Amortized Complexity
BC Commit and **BC Unveil**	$O(s)$
BC Parity	$O(sm)$
OCE	$O(sm)$
MP bit initialization	$O(sn)$
DABC	$O(sn)$
MP NOT	$O(sn)$
MP AND	$O(sn)$
MP	$O(snm)$

Although we do not have any formal proof of the optimality of our protocols, the fact that they are linear in each parameter seems to indicate that except for the constants (which are already quite small), no further complexity improvement could be achieved.

6 Acknowledgements

Special thanks to Anne Broadbent for proof reading. We are also grateful to Jörn Müller-Quade for pointing out some interesting work. This research was made possible by generous funding from Canada's NSERC.

References

[1] Beaver, D. Commodity-based cryptography (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (1997), pp. 446–455.

[2] Beaver, D. Server-assisted cryptography. In *Proceedings of the 1998 New Security Paradigms Workshop* (1998), pp. 92–106.

[3] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for noncryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), pp. 1–10.

[4] Cachin, C., Crépeau, C., and Marcil, S. Oblivious transfer with a memory bounded receiver. In *Proceedings of IEEE Symposium on Foundations of Computer Science* (1998), pp. 493–502.

[5] Cachin, C., and Maurer, U. Unconditional security against memory-bounded adversaries. In *Advances in Cryptology - CRYPTO'97* (1997), pp. 292–306.

[6] Chaum, D., Crépeau, C., and Damgård, I. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), pp. 11–19.

[7] Cleve, R. Controlled gradual disclosure schemes for random bits and their applications. In *Advances in Cryptology - CRYPTO'89* (1989), pp. 573–588.

[8] Cramer, R., Damgaard, I., Dziembowski, S., Hirt, M., and Rabin, T. Efficient multi-party computations with dishonest majority. In *EUROCRYPT* (1999), vol. 1592, Springer, pp. 311–326.

[9] Cramer, R., Damgård, I., and Maurer, U. Efficient general secure multi-party computation from any linear secret-sharing scheme. In *CRYPTO* (2000), vol. 1807, Springer, pp. 316–334.

[10] Crescenzo, G. D., Ishai, Y., and Ostrovsky, R. Universal service-providers for database private information retrieval. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing* (1998), pp. 91–100.

[11] Crépeau, C. Efficient cryptographic protocols based on noisy channels. In *Proceedings of EUROCRYPT '97* (1997), pp. 306–317.

[12] Crépeau, C. Commitment. In *Encyclopedia of Cryptography and Security* (2005), H. C. van Tilborg, Ed., vol. 12, pp. 83–86.

[13] Crépeau, C., Graaf, J., and Tapp, A. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology - CRYPTO'95* (1995), pp. 110–123.

[14] Crépeau, C., and Kilian, J. Achieving oblivious transfer using weakened security assumptions. In *Proceedings of IEEE Symposium on Foundations of Computer Science* (1988), pp. 42–52.

[15] Crépeau, C., Morozov, K., and Wolf, S. Efficient unconditional oblivious transfer from almost any noisy channel. In *Proceedings of Fourth Conference on Security in Communication Networks* (2004), pp. 47–59.

[16] Du, W., Han, Y. S., and Chen, S. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 4th SIAM International Conference on Data Mining* (2004), pp. 222–233.

[17] Du, W., and Zhan, Z. Building decision tree classifier on private data. In *Proceedings of the IEEE ICDM Workshop on Privacy, Security and Data Mining* (2002), pp. 1–8.

[18] Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (1987), pp. 218–229.

[19] Kilian, J. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), pp. 20–31.

[20] Kilian, J. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (1992), pp. 723–732.

[21] Nascimento, A. C. A., Müller-Quade, J., Otsuka, A., Hanaoka, G., and Imai, H. Unconditionally non-interactive verifiable secret sharing secure against faulty majorities in the commodity based model. In *Applied Cryptography and Network Security, Second International Conference, ACNS 2004* (2004), vol. 3089 of *Lecture Notes in Computer Science*, Springer, pp. 355–368.

[22] Rabin, T., and Ben-Or, M. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21th Annual ACM Symposium on Theory of Computing* (1989), pp. 73–85.

[23] Rivest, R. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *Unpublished manuscript* (1999).

[24] Yao, A. Protocols for secure computations. In *Proceedings of IEEE Symposium on Foundations of Computer Science* (1982), pp. 160–164.

[25] Yao, A. How to generate and exchange secrets. In *Proceedings of IEEE Symposium on Foundations of Computer Science* (1986), pp. 162–167.