Committed Oblivious Transfer and Private Multi-Party Computation

Claude Crépeau   Jeroen van de Graaf   Alain Tapp

Université de Montréal
Département d'informatique et de Recherche Opérationnelle
email:{crepeau, jeroen, tappa}@iro.umontreal.ca

Abstract. In this paper we present an *efficient* protocol for "Committed Oblivious Transfer" to perform oblivious transfer on committed bits: suppose *Alice* is committed to bits a_0 and a_1 and *Bob* is committed to b, they both want *Bob* to learn and commit to a_b without *Alice* learning b nor *Bob* learning $a_{\bar{b}}$. Our protocol, based on the properties of error correcting codes, uses Bit Commitment (BC) and one-out-of-two Oblivious Transfer (OT) as black boxes. Consequently the protocol may be implemented with or without a computational assumption, depending on the kind of BC and OT used by the participants. Assuming a Broadcast Channel is also available, we exploit this result to obtain a protocol for Private Multi-Party Computation, without making assumptions about a specific number or fraction of participants being honest. We analyze the protocol's efficiency in terms of BCs and OTs performed. Our approach connects Zero Knowledge proofs on BCs, Oblivious Circuit Evaluation and Private Multi-Party Computations in a conceptually *simple* and *efficient* way.

1 Introduction

Committed Oblivious Transfer (COT) is the natural fusion of one-out-of-two Oblivious Transfer (OT) [13] and Bit Commitment (BC). At the start of the protocol *Alice* is committed to bits $\boxed{a_0}$, $\boxed{a_1}$ and *Bob* to bit $\boxed{b}$. At the end *Bob* is committed to $\boxed{a_b}$ and knows nothing about $a_{\bar{b}}$. *Alice* learns nothing about b.

The current paper presents an *efficient* COT protocol. This protocol makes no assumption on the type of BCs and OTs that are used. For instance, with OT and BC based on the Quantum Channel [2, 3] we can perform COT without any computational assumption. Our protocol uses some elements of coding theory and simple Zero-Knowledge sub-protocols. It uses $O(n)$ OTs and $O(n^2)$ BCs, where n denotes the security parameter, but uses only $O(n)$ BCs if they have a special XOR-property. The global running time is $O(n^2)$ in the first case and $O(n)$ in the second (excluding the time necessary to build the code).

COT is a very powerful tool that can be used to perform general cryptographic tasks such as Oblivious Circuit Evaluation (OCE)[19] or Mental Games [16, 17], and Distributed Computation [18]. Such tasks have been achieved before [19] regardless of COT from generic BC and OT, but unfortunately the solution was not only complicated but very inefficient. At EUROCRYPT '89, Crépeau [8] introduced COT under the label "Verifiable Oblivious Transfer" and used it in a simpler protocol for OCE based on the work of Goldreich and Vanish [17]. Unfortunately this protocol for COT used $\Omega(n^3)$ OTs, which is still rather inefficient.

An apparently more efficient protocol for COT (using only $O(n)$ OTs) was presented in [18] under the label "Preprocess-Oblivious-Transfer". Unfortunately, it is very easy to misbehave in that protocol in a way that allows *Bob* to get both bits with non-negligible probability and for *Alice* to learn b with a non-negligible probability. These problems can be fixed easily by straightforward techniques, but then the resulting protocol becomes more or less equivalent to that of [8] using $\Omega(n^3)$ OTs. The best that we were able to get from protocols in the spirit of [8] and [18] is a protocol using $\Omega(n^2)$ OTs.

In the second part of this paper we use the COT protocol to obtain an efficient protocol for Private Multi-Party Computation (PMPC). The problem of computing a function when the participants want to keep their input private has been considered by many researchers under different models and various names (for an excellent overview see [15]). Distinguishing features of the several models are: the number of participants ($P = 2$ vs $P \geq 3$), the presence or absence of some unproven intractability assumption, the communication model (private channels between each pair of participants, broadcast channels, oblivious transfers or a combination of these) and the capacity of dealing with malicious participants.

We assume that a Broadcast Channel and an Oblivious Transfer Channel are available between each pair of participants. A protocol is a *correct* Multi-Party Computation if the output of the protocol to each participant is the same as that of the function it is emulating. Furthermore we say that such a protocol is *fair* if the the fact that one participant learns the output implies that every participant does. It is *honest* if an honest participant knows when he does not learn the output of the function and it is *private* if no coalition of less than P participants can learn information about the private inputs of other participants, other than what the output of the function logically implies.

Our result can now be stated as follows. Suppose that P participants wish to evaluate a boolean circuit F consisting of m boolean gates. Then there exists a *correct*, *private*, *honest* and *fair* protocol to evaluate F using $O(P^2n^2m)$ BC s and $O(P^2nm)$ OT s. If instead of having BC and OT as basic primitives we only have the simpler Oblivious Transfer of Rabin [26], then the number of such primitives required is $O(P^2n^3m)$.

It is rather unfortunate that a single dishonest participant can make the PMPC protocol abort. Nevertheless, this is unavoidable, since to guarantee that no dishonest coalition can have any advantage over an honest participant, the protocol requires everybody's cooperation. In section 5 of [18], Goldwasser and Levin describe a protocol (henceforth called GL) under the same assumptions as ours. Though we adopt many of their ideas (some of which are common knowledge), the protocol presented here is far more efficient. For instance, to implement an AND gate for P participants GL uses computations on polynomials, and many conditions that have to be verified while the main protocol is in progress are verified through rather inefficient sub-protocols. In this paper we demonstrate a conceptually much simpler way to accomplish this same task using and extending techniques introduced in [21, 16].

This also allows us to give a proper analysis of the complexity (in terms of the underlying cryptographic primitives) of our protocol, which in GL is quite difficult. In other words, whereas GL shows that a protocol for PMPC exists under the current assumptions, we provide a much simpler and more efficient implementation.

The remainder of this paper starts with a section elaborating on the assumptions, and introducing some useful notations. Section 3 contains the protocol and proof for COT, whereas Section 4 extends the tolls of Section 2 and finally Section 5 outlines the protocol and proof for PMPC.

2 Preliminaries

2.1 One-out-of-two Oblivious Transfer and Bit Commitment

In this paper OT and BC are used as black boxes. Since these are well-known protocols, we only describe them briefly.

In a one-out-of-two Oblivious Transfer *Bob* has to choose between learning bit a_0 or a_1 prepared by *Alice* but she does not learn his choice b. *Bob* learns a_b and obtains no information about $a_{\bar{b}}$. Implementations of OT can only exist under some assumption. For instance, OT can be constructed if trapdoor functions exist [16], from a noisy channel [11, 12], or from a quantum channel [2, 10]. It is also a well-known fact that using $O(n)$ of Rabin's Oblivious Transfers [26] one can construct one-out-of-two Oblivious Transfer [7].

In a Bit Commitment *Alice* sends a committed bit a to *Bob* in such a way that she is able to reveal it later in a *unique* way (a) but *Bob* is *not able to find* its value by himself. *Alice* cannot change her mind and open a as $\bar{a}$.

BC is impossible without making an assumption. It is easy to convert any version of Oblivious Transfer into a BC. Using error-correcting codes this is done at a cost of $O(n)$ OTs per BC [3]. BC can also be implemented under the assumption of the existence of a one-way function (or equivalently a pseudo-random bit generator) by a result of Naor [25], from a noisy channel [12], or from a quantum channel [3].

2.2 Bit Commitments with XOR

In this subsection we show how to prove that some BCs satisfy an XOR-relation without giving away their values. For this purpose we use special Bit Commitments (BCX) and proof techniques described by Kilian [21][22] (partly attributed to Rudich and Bennett). To commit to b using a BCX $\boxed{b}$, *Alice* uses $2n$ plain BCs, $(b_{1L}, b_{1R}, ..., b_{nL}, b_{nR})$ such that for each $i \in \{1...n\} : b_{iL} \oplus b_{iR} = b$. To open $\boxed{b}$, *Alice* opens its $2n$ plain BCs. *Bob* accepts only if indeed all pairs of BCs XOR to the same value b.

If *Alice* is committed to several BCXs $\boxed{b^1}, \boxed{b^2}, ..., \boxed{b^k}$ she can prove to *Bob* that $\bigoplus_{j=1}^{k} \boxed{b^j} = c$ for some public value c without disclosing the b^j's as follows. First, *Bob* specifies to *Alice* k random permutations to shuffle the n pairs b^j_{iL}, b^j_{iR} of each BCX. Then, for each $i \in \{1...n\}$ *Alice* announces $c_{iL} = \bigoplus_{j=1}^{k} b^j_{iL}$ and $c_{iR} = \bigoplus_{j=1}^{k} b^j_{iR}$. Finally, for each $i \in \{1...n\}$ *Bob* randomly asks *Alice* to open either all the $\boxed{b^j_{iL}}$'s or the $\boxed{b^j_{iR}}$'s, and verifies that the revealed values are consistent with c_{iL} or c_{iR}. If this is the case, *Bob* is convinced that the BCXs satisfy the linear relation announced by *Alice*, since her probability of convincing him of a bad relation without detection is exponentially small in n. Note that this technique can be used to prove that two BCXs $\boxed{a}$ and $\boxed{b}$ are equal (different) simply by proving $\boxed{a} \oplus \boxed{b} = 0$ ($\boxed{a} \oplus \boxed{b} = 1$).

Since each such proof destroys the BCX involved, we must copy them first. Suppose *Alice* is committed to $\boxed{b}$ and she wants two instances of this commitment, i.e. $\boxed{b^1}$ and $\boxed{b^2}$ such that $b = b^1 = b^2$. *Alice* creates $3n$ pairs of BCs such that each pair XORs to b. Then *Bob* randomly partitions these $3n$ pairs in three subsets of n pairs, thus obtaining $\boxed{b^0}, \boxed{b^1}$ and $\boxed{b^2}$ and asks *Alice* to prove that $\boxed{b^0} = \boxed{b}$ as suggested above. This destroys $\boxed{b^0}$ and $\boxed{b}$, but if *Alice* succeeds *Bob* is convinced that $\boxed{b^1}$ and $\boxed{b^2}$ are two BCXs with the same value as $\boxed{b}$.

Because of the complexity of these constructions, for the remaining of this paper we consider as a unitary BCX operation: creation of a BCX, opening of a BCX, or proof that a constant number of BCXs satisfy a given linear relation.

3 Committed Oblivious Transfer

Suppose that *Alice* is committed to bits $\boxed{a_0}, \boxed{a_1}$ and *Bob* is committed to bit $\boxed{b}$. After running COT($\boxed{a_0}, \boxed{a_1}$)($\boxed{b}$) *Bob* will be committed to $\boxed{a} = a_b$. *Alice*, whatever she does, cannot use the protocol to learn information on b and *Bob*, whatever he does, cannot use the protocol to learn information on $a_{\bar{b}}$.

3.1 Coding Theory

In our protocol for COT, a code is required. The code is not used to correct transmission errors (although it could if necessary) but in a more elaborate way for efficiency and security reasons.

Let σ and ϵ be some positive constants such that *Bob* can choose an $[n, k, d]$ linear code for which $k > (1/2 + 2\sigma)n$ and $d > \epsilon n$. The code must be efficiently decodable. The number of errors corrected by the decoding algorithm affects the efficiency of the protocol. It suffices that the algorithm corrects $\Omega(n)$ errors to guarantee the asymptotic security of the protocol.

Although Concatenated codes [14] are the most common codes with these properties, we recommend using the *Superconcentrator Codes* of Spielman [27] for their remarkable efficiency: they can be coded and decoded in linear time. In our protocol, we construct codewords, decode codewords and prove that some words are codewords. All these operations can be achieved in time $O(n)$. In particular, to prove that a committed word is a codeword it is sufficient to prove a linear number of statements each involving only a constant number of committed bits XORed together. For more details on coding theory in general we refer the reader to [24].

3.2 Informal Protocol

Intuitively $\text{COT}(\boxed{a_0}, \boxed{a_1})(\boxed{b})$ is performed by doing an imperfect "One-out-of-two Committed Oblivious String Transfer" of two random codewords c_0 and c_1. This transfer is imperfect in the sense that even the honest *Bob* learns information about both words. He learns all the bits of c_b and some bits of $c_{\bar{b}}$. We use this trick because *Bob* is not only interested in getting a_b and committing to $\boxed{a} = a_b$, but he also wants to know that if he had ran the protocol with $\bar{b}$ it would have worked as well. This is to prevent *Alice* from misbehaving on some part of her protocol so to learn b as a result of *Bob* aborting or not.

In the beginning *Alice* is committed to $\boxed{a_0}$ and $\boxed{a_1}$ and *Bob* to $\boxed{b}$. *Bob* chooses a code with suitable properties. *Alice* commits to $\boxed{c_0}$ and $\boxed{c_1}$, two randomly chosen codewords, and proves that this is the case. She Obliviously Transfers c_0 and c_1 to *Bob* in such a way that for each pair of bits *Bob* can choose to learn one of c_0 or c_1, but not both. *Bob* reads the bits to learn mostly c_b and a small part of $c_{\bar{b}}$. As suggested above he must gain a little bit of information on $c_{\bar{b}}$ to prevent *Alice* from learning which word he is interested in (and thus b). A cheating *Alice* could use very different words c_0, c_1 in the OTs and in the BCs. To prevent this, *Alice* is forced to open a small fraction of the committed bits on both sides, at *Bob*'s choosing, to let him check that the bits he received through the OT are the same. If *Alice* cheats a lot for one of the two words (or both) she will be caught without *Bob* revealing which one he was interest in. If *Alice* passes the test, only a small number of inconsistencies can remain between the bits *Bob* received and the bits *Alice* is committed to. *Bob* uses the fact that his word w should be a codeword to correct these.

Bob commits to $\boxed{w} = c_b$ and proves to *Alice* that it is a codeword. It is now *Bob*'s turn to convince *Alice* that he is committed to what he actually received through the OT (he is not committed to an arbitrary codeword, but actually to c_b). *Alice* opens a random set of positions and *Bob* shows that the bits he is committed to are consistent with the bits *Alice* reveals and $\boxed{b}$. This is sufficient to convince *Alice* that *Bob* is committed to c_b without her learning b.

Although *Bob* is committed to $\boxed{w} = c_b$, he gets some information on $c_{\bar{b}}$ revealed by the fact that it is a codeword and by the bits opened by *Alice* in the checks. To get rid of that information, *Alice* chooses a random privacy amplification [1] function h such that $a_b = h(c_b)$ and $a_{\bar{b}} = h(c_{\bar{b}})$ and proves these two relations to *Bob*. *Bob* commits to $\boxed{a} = h(w)$ and proves this to *Alice*.

3.3 Formal protocol

Before starting the protocol, *Alice* is committed to $\boxed{a_0}$, $\boxed{a_1}$ and *Bob* to $\boxed{b}$. After the protocol *Bob* will be committed to $\boxed{a_b}$. Let σ, ϵ be two positive constants as defined in Section 3.1, and let n denote the security parameter. At any point in the protocol, a failure in a proof or check leads the participants to abort.

Protocol 3.1 (COT $(\boxed{a_0},\boxed{a_1})$ $(\boxed{b})$)

1: *Bob* chooses and announces to *Alice* a decodable $[n, k, d]$ linear code C with $k > (1/2+2\sigma)n$ and $d > \epsilon n$, efficiently decoding $t \in \Omega(n)$ errors.
2: *Alice* randomly picks $c_0, c_1 \in C$, commits to $\boxed{c_0^i}$ and $\boxed{c_1^i}$, for $i \in \{1 \dots n\}$, and proves that $\boxed{c_0^1}\boxed{c_0^2}\dots\boxed{c_0^n} \in C$ and $\boxed{c_1^1}\boxed{c_1^2}\dots\boxed{c_1^n} \in C$.
3: *Bob* randomly picks $I_0, I_1 \subset \{1...n\}$, with $|I_0| = |I_1| = \sigma n, I_1 \cap I_0 = \emptyset$ and sets $b^i \leftarrow \bar{b}$ for $i \in I_0$ and $b^i \leftarrow b$ for $i \notin I_0$.
4: *Alice* runs $\text{OT}(c_0^i, c_1^i)(b^i)$ with *Bob* who gets w^i, for $i \in \{1, 2, ..., n\}$.
5: *Bob* tells $I = I_0 \cup I_1$ to *Alice* who opens $\boxed{c_0^i}$ and $\boxed{c_1^i}$, for each $i \in I$.
6: *Bob* checks that $w^i = c_{\bar{b}}^i$, for $i \in I_0$ and that $w^i = c_b^i$, for $i \in I_1$, sets $w^i \leftarrow c_b^i$, for $i \in I_0$, corrects w using C's decoding algorithm, commits to $\boxed{w^i}$, for $i \in \{1 \dots n\}$, and proves that $\boxed{w^1}\boxed{w^2}\dots\boxed{w^n} \in C$.
7: *Alice* randomly picks and announces a subset $I_2 \subset \{1...n\}$ with $|I_2| = \sigma n, I_2 \cap I = \phi$ and opens $\boxed{c_0^i}$ and $\boxed{c_1^i}$, for $i \in I_2$.
8: *Bob* proves that $\boxed{w^i} = c^i_{\boxed{b}}$, for $i \in I_2$.
9: *Alice* randomly picks and announces a privacy amplification function $h : \{0,1\}^n \to \{0,1\}$, such that $a_0 = h(c_0)$ and $a_1 = h(c_1)$ and proves $\boxed{a_0} = h\left(\boxed{c_0^1}\boxed{c_0^2}\dots\boxed{c_0^n}\right)$ and $\boxed{a_1} = h\left(\boxed{c_1^1}\boxed{c_1^2}\dots\boxed{c_1^n}\right)$.
10: *Bob* sets $a \leftarrow h(w)$, commits to $\boxed{a}$ and proves $\boxed{a} = h\left(\boxed{w^1}\boxed{w^2}\dots\boxed{w^n}\right)$.

3.4 Zero-Knowledge proofs

In the protocol, *Alice* and *Bob* make a number of zero-knowledge proofs. All these proofs are easily achieved if we are able to perform XOR proofs on committed bits. This is why the protocol uses BCXs and not plain BCs.

In Step 2 and 6 a party must show to the other that a set of committed bits form a codeword. In any linear code, this operation can be achieved by showing that the syndrome of the word is the zero-vector. This takes $O(n^2)$ BCX operations. In the case of Superconcentrator Codes, it is sufficient to prove that each bit is the XOR of a constant number of (publicly known) other bits of the word. Therefore it takes only $O(n)$ BCX operations.

In Step 8 *Bob* must prove to *Alice* that $\boxed{w^i} = c^i_{\boxed{b}}$, for $i \in I_2$. First, for each position in which $c_0^i = c_1^i$, *Bob* can simply open $\boxed{w^i}$ because in this case w^i contains no information on b. For the other positions, where $c_0^i \neq c_1^i$, one of c_0 or c_1 must be 0 and the other must be 1. When $c_0 = 0$ and $c_1 = 1$ we have $c_b^i = b$, therefore Bob proves $\boxed{w^i} \oplus \boxed{b} = 0$. In the opposite situation $c_b^i = \bar{b}$, therefore

Bob proves $\boxed{w^i} \oplus \boxed{b} = 1$. Both of these proofs give no information on b and take constant time. The total for all the positions in I_2 is $O(n)$ BCX operations.

In Step 9 and 10 a party must show to the other that $z = h\left(\boxed{x^1}\boxed{x^2}\cdots\boxed{x^n}\right)$ for some committed bit z and word x. If h is a random linear function specified by a public random subset H of the bits of x it defines a universal hash function [4]. In this case it suffices to show $z \oplus \bigoplus_{i \in H} x^i = 0$ which takes $O(n)$ BCX operations.

3.5 Validity of the Protocol

There are several ways in which *Alice* and *Bob* may misbehave when they execute our protocol. At any point *Alice* or *Bob* can decide to stop cooperating and they can commit to values different from those used in the oblivious transfer. The main point is that *Bob* cannot claim to be committed to a_b if he is not, and that even if the protocol aborts no unintended information leaks.

Bob cannot commit to $\overline{a_b}$. First observe that we cannot force *Bob* to commit to a_b after he has learned this value in Step 9. However, this is equivalent to the situation in which *Bob* refuses to open $\boxed{a}$ after the protocol since an unopened $\boxed{a}$ is useless to *Alice*. So the only important fact is that if the protocol has completed without complaints and *Bob* is committed to a bit, this must be a_b.

We show informally why this is true. Suppose *Bob* commits to $\boxed{a} = \overline{a_b}$. This means that he has proved at Step 10 that $\overline{a_b} = h(w)$. Since $a_b = h(c_b)$ this implies that $w \neq c_b$. *Bob* has proved at Step 6 that w is a codeword, so the Hamming distance between w and c_b is greater than ϵn or, equivalently, they differ in more than ϵn positions. But *Alice* has asked *Bob* to open σn positions chosen at random and for all these positions the bits were equal to their counterparts in c_b. This cannot happen except with a probability exponentially small in n.

Alice learns nothing about b. It is not hard to see that if *Alice* performs COT honestly she learns nothing about b. Furthermore, regardless of how *Alice* cheats, she gains no information about b. A straightforward analysis of each step of the protocol shows that only in Step 1, 5, 6, 8 and 10 *Bob* sends actual information to *Alice*. In Step 1 and 5 the information is independent of b. For Step 6,8 and 10 the definition of BC and the fact that the proofs are Zero-knowledge guarantee that *Alice* learns nothing about b. The only other way *Alice* could learn information would be to misbehave in such a way that *Bob* has to abort if $b = 0$ (or $b = 1$) in Steps 7 or 9. This way she could learn b and *Bob* would catch her only if $b = 1$ (or $b = 0$).

In Step 10 *Bob* will always succeed since he calculates a himself. If *Bob* fails the proofs at Step 6 or 8, this implies that $\boxed{w^1}\boxed{w^2}\ldots\boxed{w^n} \neq \boxed{c_b^1}\boxed{c_b^2}\ldots\boxed{c_b^n}$. But since c_b is a codeword (as proved in Step 2) this means that their Hamming distance is at least t because otherwise the decoding algorithm would have led to c_b. Since $t \in \Omega(n)$ the check of Step 6 has only an exponentially small probability of success. Thus, if the check of Step 6 succeeds, the proofs of Step 6 and 8 will fail with exponentially small probability whatever b is. And since $|I_0| = |I_1|$ the check of Step 6 has the same probability of success whatever b is.

Bob learns nothing about $a_{\bar{b}}$. *Alice* does not want a dishonest *Bob* to learn $a_{\bar{b}}$. In fact it is not sufficient that *Bob* learns no information about $a_{\bar{b}}$. We want that for all dishonest *Bob'* the amount of information he has about $a_{\bar{b}}$ after interacting with *Alice* will be the same as before, even if he knew a_b to start with. This is to ensure that *Bob* does not learn any correlation between a_0 and a_1 through our protocol, for instance $a_b \oplus a_{\bar{b}}$.

Note that the information on a_0 and a_1 is completely uncorrelated. Because *Alice* picks c_0 and c_1 at random, no information about $a_{\bar{b}}$ is sent before she reveals h. For *Bob* to learn h he must pass *Alice*'s test of Step 8 and convince her that he is committed to $\boxed{w} = c_b$. If *Bob* wishes to have no more than $2^{\sigma n/2}$ candidates for a_b by the time he commits at Step 6, he must have acquired on side b at least $n/2 + 3\sigma n/2$ of the $n + 2\sigma n$ bits available to him through Step 4 and 5. This is the case because there are $2^{n/2+2\sigma n}$ codewords, and thus after learning $n/2 + 3\sigma n/2$ extra bits, $2^{\sigma n/2}$ of them remain equally possible. If *Bob* gets $n/2 + 3\sigma n/2$ bits on side b, it leaves him with no more than $n/2 + \sigma n/2$ bits on side $\bar{b}$ of the $n + 2\sigma n$ bits available. In this case there are also $2^{\sigma n/2}$ candidates for $c_{\bar{b}}$ even after learning σn extra bits at Step 7.

In conclusion, if *Bob* gains as few as $n/2 + 3\sigma n/2$ bits on side b, he has probability no more than $2^{-\sigma n/2}$ of committing to the right codeword at Step 6, and any wrong codeword will have at least ϵn differences with the correct one. In this case the test of Step 8 would have an exponentially small probability in n of succeeding. On the other hand, if *Bob* gains as few as $n/2 + \sigma n/2$ bits on side $\bar{b}$ then, by a theorem of Bennett, Brassard and Robert [1], his probability of obtaining any information whatsoever about $h(c_{\bar{b}})$ is less than $2^{-\sigma n/2}$ provided h is chosen at random.

3.6 Complexity of the protocol

The exact complexity of the protocol depends on the kind of BCs and OTs used by the participants. We thus make our analysis based on the number of BCs and OTs that are required to perform the protocol. If the OT and BC being used require a constant number of communication rounds, then all the OTs and proofs can be done in parallel and the COT protocol also requires a constant number of rounds.

The protocol uses $O(n)$ OTs which are performed in Step 2, when c_0 and c_1 are transferred. It also requires $O(n)$ BCXs for both *Alice* and *Bob* to commit to c_0,c_1 resp. w. All the proofs on the BCXs can be done with only $O(n)$ BCX operations.

If the BCXs are not available one can use the trick described in Section 2.2. To perform COT, $O(n^2)$ BCs are used to perform the $O(n)$ necessary BCX operations. In this case, the overall complexity is $O(n)$ OTs and $O(n^2)$ BC operations.

4 Extension of the primitives to multiple participants

The second goal of this paper is to describe a protocol for Multi-Party Computation. To reach this goal, we enhance BCX and COT (prefixed with G for "global"), allowing other participants (who do not provide an input) to act as "verifier" to check that the active participant(s) behave honestly. Secondly we show how COT can be used to obtain a protocol for evaluating a partial AND gate between two participants, a mandatory step for PMPC.

4.1 Global Bit Commitment with XOR proof (GBCX)

A Global Bit Commitment with XOR (GBCX) is a bit commitment to a *group* of participants. A GBCX is conceptually equivalent to $P-1$ BCXs, one to each participant, all with the same value. GBCXs are constructed such that a dishonest participant $\mathcal{A}$ cannot open it as a different value to two honest participants $\mathcal{B}_j, \mathcal{B}_k$. An obvious, but very inefficient way to achieve this is that $\mathcal{A}$ proves to each pair of participants that the two BCXs they hold are equal.

We show how to do this more efficiently. To commit to a bit a using a GBCX, $\mathcal{A}$ makes $2n(P-1)$ pairs of (plain) BCs, $2n$ to each of the other participants $\mathcal{B}_1 ... \mathcal{B}_{P-1}$, such that for $i \in \{1...2n\}$ and $j \in \{1...(P-1)\}$, $a^j_{iL} \oplus a^j_{iR} = a$. Then each $\mathcal{B}_j$ chooses and broadcasts a random permutation of size $2n$ and renames his $2n$ pairs accordingly. For $i \in \{1...n\}$ $\mathcal{A}$ announces $V^0_i \leftarrow \{j : (a^j_{iL}, a^j_{iR}) = (0, a)\}$ and $V^1_i \leftarrow \{j : (a^j_{iL}, a^j_{iR}) = (1, \bar{a})\}$ in a random order. Then all participants flip n fair coins together (requiring $O(nP)$ BCs) to construct a random tuple $(S_1...S_n) \in \{L, R\}^n$. For $i \in \{1...n\}$, $j \in \{1...P-1\}$ $\mathcal{A}$ opens $\boxed{a^j_{iS_i}}$. Finally, each $\mathcal{B}_j$ verifies that the values of the $a^j_{iS_i}$ opened to him correspond with those broadcasted, and that $W_i \leftarrow \{j : a^j_{iS_i} = 0\}$ equals either V^0_i or V^1_i.

After this protocol $\mathcal{A}$ has n pairs of untouched BCs with each $\mathcal{B}_j$ (those with $i \in \{(n+1)...2n\}$), which constitute the new GBCX. It is easy to verify that if $\mathcal{A}$ tries to commit to two different bits with two honest participants, or if he tries to construct inconsistent commitments, he will be caught by each pair of honest participants except with exponentially small probability.

Although the role of $\mathcal{A}$ (the "*active*" participant) is different from the role of the $\mathcal{B}_j$s (the "*passive*" participants) in the protocol, the cost of creating a GBCX is the same for everybody, $O(Pn)$ BCs. Once created, operations on GBCXs, such as proving linear relations and copying, are reduced to P operations between $\mathcal{A}$ and $\mathcal{B}_j$ using BCXs. For $\mathcal{A}$ the cost of one such operation is $O(Pn)$, but for the $\mathcal{B}_j$ it is only $O(n)$ (since after the creation they do not need to interact with other users).

4.2 Global Committed Oblivious Transfer (GCOT)

Global Committed Oblivious Transfer (GCOT) is the extension of COT to a group. For *Alice* and *Bob* GCOT achieves exactly the same functionality as COT but it allows the passive participants to be convinced that *Alice* and *Bob* do not conspire, i.e. that indeed after GCOT($\boxed{a_0}$,$\boxed{a_1}$)($\boxed{b}$) *Bob* is committed to $\boxed{a_b}$.

Some modifications are necessary to change COT to GCOT: (1) The error-correcting code C of Step 1 must be chosen by all participants; (2) BCXs are replaced by GBCXs and all commitment openings and proofs are done to each participant; (3) the subset I_2 in Step 7 must be chosen by all participants.

In order to choose I_2, only $O(n \log(n))$ random coins have to be flipped, at the expense of $O(Pn \log(n))$ plain BCs for each participant. When many GCOTs on different inputs are performed in parallel, as will be the case in the evaluation of AND gates in the final protocol, the same I_2 may be used. Therefore we do not take this cost into account in the next paragraph, but deal with it later.

For one GCOT *Alice* and *Bob* have to perform $O(n)$ *active* GBCX operations (resulting in a total of $O(Pn^2)$ BCs) and $O(n)$ OTs. The others have to perform $O(n)$ *passive* GBCX operations (resulting in $O(n^2)$ BCs).

4.3 PAND and GPAND

As a step towards PMPC we introduce a two-party protocol, called PAND, that takes the BCXs $\boxed{a}$ from *Alice* and $\boxed{b}$ from *Bob* as inputs. After the execution of PAND *Alice* is committed to $\boxed{a'}$ and *Bob* to $\boxed{b'}$ such that $a \wedge b = a' \oplus b'$, and neither participant learns the other's input value. Since the output is composed of the pair a' and b', we call this protocol PAND for Pair-AND.

We implement PAND using COT. *Alice* randomly chooses a bit a', commits to $\boxed{a''} = \boxed{a'} \oplus \boxed{a}$ and proves it. Then *Alice* performs COT($\boxed{a'}$,$\boxed{a''}$)($\boxed{b}$) with *Bob* who gets $\boxed{b'}$. (Notice that $b = 0 \Rightarrow b' = a' \Rightarrow a' \oplus b' = 0$ and that $b = 1 \Rightarrow b' = a' \oplus a \Rightarrow a' \oplus b' = a$, so $a \wedge b = a' \oplus b'$.)

GPAND is a generalization of the PAND to a group: two active participants want to make a PAND, while the passive participants want to be sure that indeed $\boxed{a} \wedge \boxed{b} = \boxed{a'} \oplus \boxed{b'}$. The GPAND is done as a PAND, with COT replaced by GCOT. The cost of one GPAND is of the same order as the cost of one GCOT, both for the active and the verifying participants.

5 Private Multi-Party Computation (PMPC)

In this section we show how, given OT and BC between each pair of participants and a reliable Broadcast Channel, P participants can perform a *correct*, *honest*, *fair* and *private* Multi-Party Computation, where we make no assumptions about the number of honest participants. The protocol consists of three steps: *initialization*, *computation* and *revelation*. At each step we make extensive use of the primitives previously defined.

5.1 Initialization

In the *initialization* step all participants agree on the circuit F to be evaluated and on a security parameter n. They also agree on a parameter σ and a code C for the GCOT sub-protocol.

In the PMPC protocol each bit involved is represented by a Distributed Bit Commitment (DBC) consisting of P shares, each share being a GBCX created by a different participant. The value of a DBC is the XOR of the value of its shares, therefore even $P-1$ participants are not able to reconstruct the value of the DBC if one participant refuses to cooperate.

As a second part of the initialization step each participant uses DBCs to commit to his input bits. To create a DBC of value a, $\mathcal{A}$ asks each participant to commit to a random bit using a GBCX, and to open it to $\mathcal{A}$ only. Then $\mathcal{A}$ creates a GBCX such that the XOR of all the GBCXs equals a.

5.2 Computation

In the *computation* step the participants evaluate the circuit consisting of AND and NOT gates, one gate after the other, where the input and output bits of each gate are DBCs.

Since the output bit of a gate can be an input bit to several other gates we must be able to copy DBCs. To make r copies of a DBC each participant makes r copies of his share (a GBCX). The NOT operation on a DBC is equivalent to a copy, except that one designated participant inverts the value of his GBCX.

To evaluate an AND gate on two DBCs of value a and b we observe that $(\bigoplus_{i=1}^{n} a_i) \wedge (\bigoplus_{j=1}^{n} b_j) = \bigoplus_{i,j=1}^{n} a_i \wedge b_j$. In other words, one Distributed AND can be reduced to P^2 GPANDs, one between each pair of participants. After all the GPANDs have been done each participant chooses a GBCX that equals the XOR of all his shares and he proves this. This GBCX is his share of the DBC whose value equals $a \wedge b$.

To evaluate one AND gate $O(P^2)$ GPANDs and so $O(P^2)$ GCOTs are executed. Each participant will be actively involved in P GCOTs, each requiring $O(n)$ OTs and $O(n)$ *active* GBCXs operations. Each participant also verifies $O(P^2)$ GCOTs between other participants. For one such GCOT verification he is involved in $O(n)$ *passive* GBCX operations. Therefore each participant is involved in $O(Pn)$ *active* GBCXs and $O(P^2 n)$ *passive* GBCXs. This results in $O(P^2 n^2)$ BCs and in $O(Pn)$ OTs.

We must also take into account the cost for creating the subset I_2 used in every GCOT. Only one random I_2 suffices for many parallel GCOTs, and since the creation of a single I_2 requires $O(n \log(n))$ coin flips, it requires a total of $O(Pn \log(n))$ BCs per participant. This does not change the previously stated complexity.

5.3 Revelation

At the end of the *computation* step each output bit of the circuit is hidden in a DBC. In order for each honest participant to learn an output bit of F, every participant could open his share of the DBC representing the answer, but obviously a dishonest participant could quit when he has more information than others. Better solutions, which achieve *fairness*, appear in the literature [6][18]. They can easily be incorporated into our protocol.

5.4 The validity of the PMPC protocol

We do not claim to provide a full proof that the protocol presented here is secure. The security of the final protocol relies on the security of all the sub-protocols that constitute it. Due to space restrictions, we only give a brief and sketchy proof that PMPC satisfies the four desired properties mentioned in the introduction.

Correctness: If all participants are honest the output of our protocol should equal the output of the function to be emulated. This is verified because each sub-protocol outputs the value it is designed for.

Privacy: Observe that from a participant's point of view the GBCX is the smallest "unit" to which he can commit and that no coalition can open it if the owner does not cooperate. Because of the way each DBC is constructed each participant holds a share of each bit of the computation, so for all sub-protocols no information about the values of the DBCs is revealed as long as at least one participant is honest. Therefore, before the revelation of the output bits no coalition of less than P participants can obtain any information whatsoever on the input bits, intermediate bits or output bits of the computation, except for what can be deduced from the information already known by the coalition. A more formal analysis along these lines could benefit from the definition of privacy given in [23].

Fairness: This is an issue in the revelation step only. We want to prevent one or several parties from learning more information on the output bits before the others. For the fairness of our protocol we totally rely on the model and definitions given by Goldwasser and Levin [18]. Though the implementation of their protocol differs from ours their model still applies, in particular their notion of fairness.

Honesty: Note that no coalition of participants can cheat on the resulting output DBCs of any sub-protocol. For the full protocol this means that, once the input bits have been committed to, no coalition of less than P players can change the values of an output DBC without being caught (except with exponentially small probability). In other words, even though any coalition can disrupt the protocol by preventing it from completing, no coalition can make an honest participant accept the output of F when any of the output DBCs has been tampered with.

5.5 Complexity of the PMPC protocol

It is easy to see that to evaluate a circuit F composed of m gates $O(m)$ DBC operations are needed. The most expensive operation on DBCs is the evaluation of an AND gate, so the overall complexity of the PMPC protocol is $O(P^2n^2m)$ BCs and $O(Pnm)$ OTs. Since BC and OT can be implemented using $O(n)$ of Rabin's Oblivious Transfers, PMPC can be performed using $O(P^2n^3m)$ of them.

Acknowledgments

We thank Joe Kilian and Matt Franklin for providing references. We give special thanks to Mélanie and Marie-France for their patience and understanding.

References

1. C.H. Bennett, G. Brassard, J.-M. Robert, *Privacy Amplification by Public Discussion*, SIAM Journal on Computing, Vol. 17, No.2, 1988, pp. 210–229.
2. C.H. Bennett, G. Brassard, C. Crépeau, M.-H. Skubiszewska, *Practical Quantum Oblivious Transfer*, Advances in Cryptology - CRYPTO'91, Springer-Verlag, 1992, pp. 351–366.
3. G. Brassard, C. Crépeau, R. Jozsa, D. Langlois, *A Quantum Bit Commitment Scheme Provably Unbreakable by both Parties*, 34th IEEE Symposium on Foundation of Computer Science, 1993, pp. 362–371.
4. J. L. Carter and M. N. Wegman, *Universal Classes of hash function*, Journal of Computer and System Sciences, Vol. 18, 1979, pp. 143-154.
5. D. Chaum, I. Damgård and J. van de Graaf, *Multiparty computations ensuring privacy of each party's input and correctness of the result*, Advances in Cryptology - CRYPTO'87, Springer-Verlag, 1988, pp. 87–119.
6. R. Cleve, *Controlled Gradual Disclosure Schemes for Random Bits and Their Applications*, Advances in Cryptology - CRYPTO'89, Springer-Verlag, 1991, pp. 573–590.
7. C. Crépeau, *Equivalence Between Two Flavours of Oblivious Transfer*, Advances in Cryptology - CRYPTO'87, Springer-Verlag, 1988, pp. 350-354.
8. Crépeau, C., *Verifiable Disclosure of Secrets and Applications*, Advances in Cryptology - Eurocrypt'89, Springer-Verlag, 1990, pp. 181–191.
9. C. Crépeau, *Correct and Private Reductions Among Oblivious Transfers*, Ph.D. thesis, MIT, 1990.
10. C. Crépeau, *Quantum Oblivious Transfer*, Journal of Modern Optics, vol. 41, No. 12, 1994.
11. C. Crépeau, J. Kilian, *Achieving Oblivious Transfer Using Weakened Security Assumptions*, 29th IEEE Symposium on Foundation of Computer Science, 1988, pp. 42–52.
12. C. Crépeau, *Cryptographic protocol based on noisy channel*, in preparation, 1995.
13. S. Even, O. Goldreich and A. Lempel, *A Randomized Protocol for Signing Contracts*, Communications of the ACM, Vol 28, 1985, pp. 637-647.
14. Forney, G. D., *Concatenated Codes*, The M.I.T. Press, 1966.
15. M. Franklin, *Complexity and security of distributed protocols*, Ph. D. thesis, Computer Science Department of Columbia University, New York, 1993.
16. O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game, or: A completeness theorem for protocols with honest majority*, 19th ACM Symposium on Theory of Computing, 1987, pp. 218–229.
17. O. Goldreich, R. Vainish, *How to solve any protocol problem — an efficiency improvement*, Advances in Cryptology - CRYPTO'87, Springer-Verlag, 1988, pp. 73–86.
18. S. Goldwasser, L. Levin, *Fair computation of general functions in presence of moral majority*, Advances in Cryptology - CRYPTO'90, Springer-Verlag, 1991, pp. 77–93.

19. J. Kilian, *Founding cryptography on Oblivious transfer*, 20th ACM Symposium on Theory of Computation, 1988, pp. 20–31.
20. J. Kilian, *Uses of Randomness in Algorithms and Protocols*, MIT Press, 1990.
21. J. Kilian, *A note on efficient zero-knowledge proofs and arguments*, 24th ACM Symposium on Theory of Computation, 1992, pp. 723–732.
22. J. Kilian, *On the complexity of bounded-interaction and noninteractive zero-knowledge proofs*, 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 466-477.
23. E. Kushilevitz, S. Micali, R. Ostrovski, *Reducibility and completeness in multi-party private computations*, 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 478-489.
24. F. J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.
25. M. Naor, *Bit Commitment using Pseudo-Randomness*, Advances in Cryptology - CRYPTO'89, Springer-Verlag, 1989, pp. 128–136.
26. M. Rabin, *How to exchange secrets by oblivious transfer*, Tech. Memo TR–81, Aiken Computation Laboratory, Harvard University, 1981.
27. D. Spielman, *Linear-Time Encodable and Decodable Error-Correcting Codes*, 27th ACM Symposium on Theory of Computing, 1995, pp. 388–397.