Appears in *Proceedings of 27th Symposium on Information Theory in the Benelux* (2006) 283–290.

A Protocol Issue for the Malicious Case of Yao's Garbled Circuit Construction

Mehmet S. Kiraz m.kiraz@tue.nl
Berry Schoenmakers berry@win.tue.nl
Dept. of Mathematics and Computer Science, TU Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Abstract

We present a protocol issue that arises with the use of oblivious transfer in the malicious case of several two-party computation protocols based on Yao's garbled circuit. We describe this issue for a protocol by Pinkas (Eurocrypt 2003) and for the Fairplay protocol, and we discuss why this issue still persists for a recently suggested modification of the Fairplay protocol. We propose a solution using committed oblivious transfer instead of (plain) oblivious transfer. We also introduce the notion of *committing oblivious transfer*, which leads to an alternative, potentially more efficient solution.

1 Introduction

The concept of general secure two-party computation was introduced by Yao together with a solution in terms of so-called garbled circuits [10, 11]. Such a garbled circuit is an encrypted form of a function $f(x, y) = (f_1(x, y), f_2(x, y))$, where $f_1(x, y)$ and $f_2(x, y)$ denote the respective outputs of the two parties, and is jointly evaluated by the two parties.

Briefly, Yao's protocol is as follows. The idea is that one party (the constructor Bob) generates the garbled circuits, and that the other party (the evaluator Alice) evaluates the garbled circuits. Assume that the function $f(x, y)$ is represented as a boolean circuit. For each wire (input/internal/output) in the boolean circuit, Bob uses two random bit (garbled) strings that are assigned to the corresponding values 0 and 1, respectively. Bob will send *only* the garbled strings corresponding to his input values to Alice. Furthermore, 1-out-of-2 oblivious transfer (OT) is used to let Bob provide Alice with *only* the garbled strings corresponding to her input values, without Bob learning which strings she gets. At the end of the evaluation, Alice needs to tell Bob the garbled strings she found for his output wires. Alice also converts the garbled strings she found for her output wires to bits (she can do this using the information obtained from Bob). We refer to [9] for a precise description of Yao's protocol. Also, Lindell and Pinkas recently gave a simulation-based proof of security for the garbled circuit construction for the passive case (honest-but-curious parties) [4].

In this paper, we discuss the malicious case of the garbled circuit construction. In particular, we will be concerned with the case of a malicious Bob, as this is the hard part. The point is that Bob may generate the garbled circuits anyway he likes, which may clearly affect the privacy of Alice's inputs and the correctness of her outputs. Protection against malicious behavior by Bob is done by using a cut-and-choose approach, where Bob will generate multiple garbled circuits (for the same function $f(x, y)$), and Alice will challenge Bob to open some subset for verification.

A basic cut-and-choose approach is the 1-out-of-m technique as used for instance in Fairplay [8]: Bob constructs m independently generated garbled circuits for the function $f(x, y)$ and sends these to Alice. Alice chooses one of these circuits uniformly at random for the evaluation step. Bob then opens all the remaining $m - 1$ circuits so that Alice can verify that these circuits are correct. Next, they run OT in order for Alice to receive her inputs in garbled form, as described above for Yao's protocol. Bob also sends the garbled strings for his inputs to Alice so that she can evaluate the chosen circuit without further interaction with Bob. Alice computes all the garbled output values of the chosen circuit and sends Bob's garbled output values to him. It follows that Bob can cheat with probability $1/m$.

Pinkas [7] proposed a protocol which uses a more advanced cut-and-choose approach, namely an $m/2$-out-of-m technique to achieve exponentially small cheating probability. Again, Bob constructs m garbled circuits and sends these to Alice. Alice now chooses $m/2$ circuits randomly for evaluation and tells Bob to open the remaining ones. She verifies whether all the opened circuits are correctly prepared. Next, they run OT in order for Alice to receive the garbled strings for her inputs to the chosen circuits. Bob also sends the garbled strings for his inputs to Alice so that she can evaluate the chosen $m/2$ circuits and compute her and Bob's garbled output values. She sends the output values of only one of the chosen $m/2$ circuits to Bob (namely, only for a so-called *majority circuit*), such that Bob cannot cheat successfully except with exponentially small probability (in m).

The problem that we address in this paper stems from the particular way OT is used in the cut-and-choose protocols mentioned above. A malicious Bob cannot simply be assumed to send the right garbled strings to Alice during the OT protocol, and we point out that without any further protection Alice's privacy and the correctness of her outputs may be affected. This protocol issue is not related to OT per se, but rather with the link between OT and the surrounding protocol. A similar point is made in the recent paper by Mohassel and Franklin [5] (of which our work is independent). Their discussion and proposed solution focuses on Fairplay [8], but as we will point out Bob will still be able to cheat in a critical way.

To eliminate cheating by Bob, we will resort to the use of more powerful (but also more costly) types of OT, namely committed OT, and a new type of OT introduced in this paper which we call *committing* OT.

2 The Protocol Issue

In this section, we first describe the issue with the use of OT in the Fairplay protocol [8], which is based on a 1-out-of-m cut-and-choose approach. The same issue with the use of OT arises in the $m/2$-out-of-m cut-and-choose approach used in [7]. Next, we show how the issue persists for the modification to the Fairplay protocol proposed by Mohassel and Franklin [5].

For our purposes the following high-level definition of oblivious transfer suffices. Note that we focus on 1-out-of-2 oblivious transfer throughout the paper.

Definition 1 (OT) 1-out-of-2 oblivious transfer is a protocol between two parties, called the sender S and the receiver R. The private input of S consists of bit strings a^0 and a^1 and the private input of R is a bit b. The private output for R is the bit string a^b, and there is no output for S (see Figure 1).

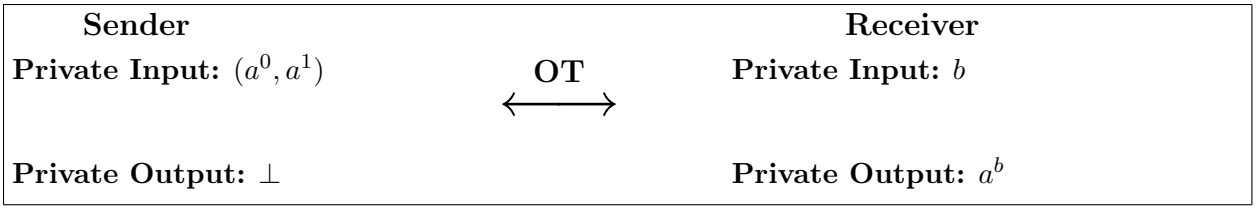

Figure 1: 1-out-of-2 Oblivious Transfer

Such an OT protocol is supposed to be secure, meaning that both correctness and privacy are ensured: only the outputs *as specified* become available to the parties and *nothing* else.

We now discuss the use of OT in Yao's garbled circuit construction. In the semi-honest (passive) case, OT can clearly be used to achieve a secure protocol for two-party computation. However, when switching to the malicious (active) case, OT should be used with much more care. In particular, it should be taken into account that, e.g., a sender may use bit strings a^0 and a^1 which are unrelated to other parts of the surrounding protocol. As we will explain next, subtle problems arise if the values used in an OT protocol are not (properly) linked to the surrounding protocol.

Recall that at some stage in Yao's protocol, OT is used to let Bob provide Alice the garbled strings corresponding to her input bits. To discuss the problem it suffices to consider a single OT in which Alice uses an input bit b as her private input, say, and Bob uses garbled strings w^0 and w^1 as his private input, where w^0 corresponds to bit value 0 and w^1 represents bit value 1. Alice can only evaluate the corresponding circuit correctly if she indeed gets the bit string w^b at the end of the OT protocol. The problem with the extensions to the malicious case of Yao's protocol as presented in [7, 8] is that a malicious Bob is not stopped from using other values than w^0 and w^1.

Therefore, Bob can follow several strategies to compromise correctness and/or privacy of Yao's protocol in the malicious case. We mention a few obvious deviations of the protocol, resulting in wrong values $\tilde{w}^0, \tilde{w}^1$ as follows:

- Bob may interchange the values of w^0 and w^1, putting $(\tilde{w}^0, \tilde{w}^1) = (w^1, w^0)$;
- Bob may duplicate either of the values w^0 and w^1, putting either $(\tilde{w}^0, \tilde{w}^1) = (w^0, w^0)$ or $(\tilde{w}^0, \tilde{w}^1) = (w^1, w^1)$;
- Bob may replace either or both of the values with a bogus value, denoted by $*$, putting $(\tilde{w}^0, \tilde{w}^1) = (w^0, *)$, or $(\tilde{w}^0, \tilde{w}^1) = (*, w^1)$, or even $(\tilde{w}^0, \tilde{w}^1) = (*, *)$.

The consequences for Alice's security are as follows in these cases.

- If Bob interchanges the values of w^0 and w^1 then clearly Alice will receive an incorrect garbled string for her input wire, but she will continue to evaluate the circuit without noticing anything (but obtaining wrong output values). Similarly, if Bob duplicates either of the values w^0 and w^1 then Bob is sure which input value Alice is using, and possibly Alice gets a wrong output (without her noticing anything).
- If Bob uses $(w^0, *)$ in OT where $* \neq w^1$ then depending on Alice's input bit b, she will either be able to evaluate the circuit and produce an output for Bob (and not notice

anything) if $b = 0$, or she will notice that she cannot evaluate the circuit and cannot produce an output for Bob if $b = 1$. Clearly, this way Bob learns the value of Alice's input b, and Alice cannot do anything about it.

To eliminate these problems, Bob must be forced to use the correct values in OT.

In the remainder of this section, we review the proposed modification for Fairplay by [5] and then present the reasons why the problem is not completely eliminated. At this point we also introduce the following notation for commitments, which we use throughout the paper.

Recall that a non-interactive commitment scheme can be described in terms of a single algorithm, usually called commit(). Accordingly, we let $\text{commit}_P(s, r)$ denote a commitment by party P to a bit string s, using random bit string (witness) r. If the witness is irrelevant, we suppress it from the notation by writing $\text{commit}_P(s)$.

For simplicity, we assume that the garbled circuit has only one input wire for Alice. The more general case of multiple input wires can be obtained by simple extension of this case. The modification proposed in [5] is as follows. When Bob sends the garbled circuits to Alice, he also generates commitments to the garbled strings w_j^0 and w_j^1, used for Alice's input wire in the j-th circuit, denoted as $\text{commit}_B(w_j^0, r_j^0)$ and $\text{commit}_B(w_j^1, r_j^1)$, respectively where r_j^0 and r_j^1 are random bit strings. When verifying the $m-1$ opened circuits, Bob is also supposed to open these commitments and Alice verifies that these commitments are correct and whether the committed garbled strings w_j^0 and w_j^1 correspond to the garbled strings used in the j-th circuit.

If verification succeeds, Alice is almost sure that the *unopened* circuit and the corresponding *unopened* commitments are correct as well. For the *unopened* circuit they run OT for Alice's input wire where Bob is the sender holding two witnesses and two garbled strings and Alice is the receiver holding her input bit. At the end of OT Alice receives one of the witnesses and the corresponding garbled string.

As a consequence, with probability at least $1 - 1/m$ Alice is ensured that the committed values for the *unopened* circuit are correct. Therefore, with probability at least $1 - 1/m$, Alice will notice when she gets a wrong garbled string for her input wire (e.g., because Bob interchanged or duplicated the garbled strings), and by using an $m/2$-out-of-m technique, this probability will be close to 1. The problem with this proposal, however, is that it is not considered what happens once a cheating Bob sends a bogus value in the OT protocol, following the same scenario as described above.

A cheating Bob will simply construct all the garbled circuits and all the commitments correctly. Therefore, Alice will certainly accept the opened circuits and the opened commitments. But during OT Bob may cheat as described above. Suppose Bob decides to use $((w_j^0, r_j^0), (*, *))$ as input to OT. Then, Alice's input bit is 0, she will obtain correct values (w_j^0, r_j^0), which pass all verifications, and she will be able to evaluate the circuit without any problems. But, if Alice's input bit is 1, then she gets bogus values, which she'll notice, and she is unable to hide this fact from Bob. Clearly, Bob is able to tell what Alice's input bit is, thereby compromising privacy.

3 An Improved Protocol

In this section, we propose two schemes to fix the issue. We first describe how we can repair the issue with the use of committed OT. Next, we define a new notion called *committing OT* in order to obtain a possibly more efficient result in the garbled circuit approach.

3.1 Using Committed Oblivious Transfer

We describe how we can fix the issue for the case 1-out-of-m technique using committed OT which works for bit strings. Committed OT is defined as a combination of bit commitment and 1-out-of-2 OT in the literature (se, e.g., [2, 3]). Similarly, we define committed OT for bit strings. Constructing efficient committed OT for bit strings is a topic, which we do not address in this paper.

Definition 2 (Committed-OT) Committed 1-out-of-2 oblivious transfer is a protocol between two parties, called the sender S and the receiver R. The private input of S consists of bit strings a^0, a^1, r^0, r^1 and the private input of R is a bit b and bit string s. The common input for S and R consists of $\mathrm{commit}_S(a^0, r^0)$, $\mathrm{commit}_S(a^1, r^1)$ and $\mathrm{commit}_R(b, s)$. The private output for R consists of the bit string a^b and a bit string t, and the common output for S and R is $\mathrm{commit}_R(a^b, t)$ (see Figure 2).

Sender	**Common Input:** $\mathrm{commit}_R(b,s)$, $\mathrm{commit}_S(a^0,r^0)$, $\mathrm{commit}_S(a^1,r^1)$	**Receiver**
Private Input: a^0, a^1, r^0, r^1		**Private Input:** b, s
	Committed OT $\longleftrightarrow$	
Private Output: $\perp$	**Common Output:** $\mathrm{commit}_R(a^b, t)$	**Private Output:** a^b, t

Figure 2: Committed Oblivious Transfer

The 1-out-of-m cut-and-choose protocol using committed OT runs as follows. Before Bob sends the garbled circuits to Alice, he also generates commitments to the garbled strings for her inputs. He then sends the garbled circuits together with the corresponding commitments to Alice. When Alice asks Bob to open the circuits he also opens all the corresponding commitments. Alice is now almost sure that the commitments for the *unopened* circuit are correct. As required for committed OT, Alice commits to her input bits. Then for the *unopened* circuit they run committed OT for every input wire of Alice from which Alice learns her corresponding garbled input values.

Now, any attempt by Bob to use other values than the correct garbled strings in committed OT will be detected by Alice, and she can abort the protocol at this point. The point is that she can abort without revealing any information on her input bits, as committed OT will only finish successfully (by definition) if Bob's private inputs correspond to the commitments. As desired, the cheating probability of Bob is thus $1/m$ (which can be made exponentially small by using committed OT in combination with $m/2$-out-of-m cut-and-choose).

3.2 Using Committing Oblivious Transfer

As described above, the use of committed OT also requires Alice to send commitments to her input bits, and one may wonder why this would be needed. In fact, the use of committed OT may be overkill for securing the cut-and-choose protocols under consideration. Therefore, we also propose to use different versions of OT, which conceptually sit between plain OT and committed OT. As a concrete example, we introduce the notion of *committing oblivious transfer*, which is potentially weaker than committed OT but also allows for potentially more efficient protocols than committed OT.

The idea for committing OT is that Bob is also supposed to send commitments to his input values used in the OT protocol, so that Alice can check for the opened circuits whether they are correct values or not.

Definition 3 (Committing-OT) Committing 1-out-of-2 oblivious transfer is a protocol between two parties, sender S and the receiver R. The private input of S consists of bit strings a^0 and a^1 and the private input of R is a bit b. The private output for S consists of bit strings r^0 and r^1, the private output for R consists of the bit string a^b, and the common output for S and R is $\text{commit}_S(a^0, r^0)$ and $\text{commit}_S(a^1, r^1)$ (see Figure 3).

Sender		**Receiver**
Private Input: (a^0, a^1)	**Committing OT** $\longleftrightarrow$	**Private Input:** b
Private Output: r^0, r^1	**Common Output:** $\text{commit}_S(a^0, r^0)$,$\text{commit}_S(a^1, r^1)$	**Private Output:** a^b

Figure 3: Committing Oblivious Transfer

Using committing OT, we now describe an alternative solution.

Before Alice chooses which circuits she wants to evaluate, Alice and Bob run committing OT for all garbled circuits. This will be done using only one committing OT per input wire of Alice. Thus Alice receives the garbled strings for her input values, together with commitments on all garbled strings associated with her input wires. She will check the commitments for the opened circuits, ensuring her that the garbled strings for the *unopened* circuits are valid as well, with high probability.

More concretely, a protocol using a 1-out-of-m technique with committing OT may run as follows.

Step 1. Bob sends garbled circuits GC_j for $j \in \{1, \ldots, m\}$ to Alice.

Step 2. For each input wire of Alice the following is done. Let x_i denote Alice's input bit for the i-th input wire. Alice and Bob run committing OT. Bob is the sender holding the bit strings $(w^0_{i,1}, \ldots, w^0_{i,m})$ and $(w^1_{i,1}, \ldots, w^1_{i,m})$ consisting of the garbled strings for Alice's input wires in all GC_j, $j \in \{1, \ldots, m\}$. Alice is the receiver holding her input bit x_i. At the end of committing OT, Alice gets $(w^{x_i}_{i,1}, \ldots, w^{x_i}_{i,m})$ .The commitments by Bob to his input strings are common output.

Step 3. Alice randomly chooses $r \in_R \{1, \ldots, m\}$ and sends r to Bob.

Step 4. Bob opens the circuits GC_j and the corresponding commitments $\text{commit}_B(w_{i,j}^b)$ for $j \in \{1, \ldots, m\} \setminus \{r\}$ and $b \in \{0, 1\}$.

Step 5. Alice first checks whether the opened circuits are correct, and then she verifies all committing OTs. If all are correct, Alice evaluates circuit GC_r as before.

The cheating probability of Bob is now $1/m$.

A difference with the protocols based on committed OT of the previous section is that committing OT is applied to all circuits, whereas committed OT is applied to the *unopened* circuits only. Note, however, that for $m/2$-out-of-m cut-and-choose protocols, the number of *unopened* circuits is exactly half of the total number of circuits. Hence, for the general case when Bob's cheating probability should be exponentially small, the protocol based on committing OT may be more efficient than one based on committed OT.

Finally, we note that the OT protocol by Bellare and Micali based on ElGamal encryption [1] can be seen as an instance of committing OT, as the sender outputs two encryptions for his respective inputs. Another instance of committing OT is presented by Naor and Pinkas which is the first two-round efficient committing OT without using random oracles [6].

References

[1] M. Bellare and S. Micali. Non-Interactive Oblivious Transfer and Applications. *In Proceedings of Crypto '89, LNCS 435*, pages 547–557, 1990.

[2] Claude Crepeau, Jeroen van de Graaf, and Alain Tapp. Committed Oblivious Transfer and Private Multi-Party Computation. *In Proceedings of Crypto '95, LNCS 963*, pages 110–123, 1995.

[3] J. Garay, P. MacKenzie, and K. Yang. Efficient and Universally Composable Committed Oblivious Transfer and Applications. *In Proceedings of First Theory of Cryptography Conference (TCC 2004), LNCS 2951*, pages 297–316, 2004.

[4] Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two–Party Computation. *Cryptology ePrint Archive, Report 2004/175*, 2004.

[5] P. Mohassel and M.Franklin. Efficiency Tradeoffs for Malicious Two-Party Computation. *In Proceedings of Public Key Cryptography Conference (PKC '06), LNCS 3958*, pages 458–473.

[6] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. *In Proceedings of SODA '01*, pages 448–457, 2001.

[7] B. Pinkas. Fair Secure Two–Party Computation. *In Proceedings of Eurocrypt 03, LNCS 2656*, pages 87–105, 2003.

[8] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella. Fairplay–A Secure Two–Party Computation System. *In USENIX Security*, pages 287-302, 2004.

[9] P. Rogaway. The Round Complexity of Secure Protocols. *PhD Thesis, MIT*, June 1991.

[10] A. C. Yao. Protocols for Secure Computation. *In Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[11] A. C. Yao. How to Generate and Exchange Secrets. *In Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 162–167, 1986.