# Efficiency Tradeoffs for Malicious Two-Party Computation

Payman Mohassel and Matthew Franklin

Department of Computer Science, University of California, Davis CA 95616
mohassel@cs.ucdavis.edu, franklin@cs.ucdavis.edu

**Abstract.** We study efficiency tradeoffs for secure two-party computation in presence of malicious behavior. We investigate two main approaches for defending against malicious behavior in *Yao's garbled circuit* method: (1) *Committed-input* scheme, (2) *Equality-checker* scheme. We provide asymptotic and concrete analysis of communication and computation costs of the designed protocols. We also develop a weaker definition of security (*k-leaked model*) for malicious two-party computation that allows for disclosure of some information to a malicious party. We design more efficient variations of *Yao's* protocol that are secure in the proposed model.

**Keywords**: secure two-party computation, secure function evaluation, Yao's garbled circuit, malicious adversary.

## 1 Introduction

General two-party secure computation was an early success of modern cryptography. *Yao's garbled circuit* protocol [Yao86] is a classic and elegant solution to this problem. Thanks to Lindell and Pinkas [LP04] (building on Goldreich [Gol04] and others), we now have a careful proof of Yao's protocol in a suitable formal framework.

It is well-known that Yao's protocol is vulnerable to malicious behavior by its participants. The classic solution to this issue is the *zero-knowledge* compilation of Goldreich et al. [GMW86,GMW87,Gol04]. This paradigm is of great theoretical interest, but is not efficient in practice. For this reason, various alternative methods for protecting Yao's protocol against malicious behavior have been suggested [Pin03,MNPS04].

The general approach is based on *cut-and-choose* techniques that tend to gain efficiency at the cost of increased risk of undetected cheating. We cite in particular, the impressive Fairplay system of Malkhi et al. [MNPS04], which has made a major step forward in bringing Yao's protocol to practice, and which was the starting point for our work.

Although these cut-and-choose ideas are intuitive and natural, they have some hidden subtleties and complexities. Indeed, we show that one of the protocols in the Fairplay paper has a subtle bug that allows one of the parties to cheat undetectably. This suggests that cut-and-choose designs for protecting Yao's protocol from malicious behavior deserve a closer look.

Another reason to take a closer look at this design space is that the tradeoffs of efficiency vs. undetected cheating are not immediately apparent (especially when combined with other cryptographic techniques). We find some nice constructions with attractive balances of the relevant parameters. We are not claiming that the best possible tradeoffs have been found. In fact, the design space is so rich that we suspect that more work remains to be done in this area. This is especially true when the parameter tradeoffs includes the number of bits of secret information leaked to a malicious party (a setting we explore in Section 4).

### 1.1 Related Work

We mention some of the work in the literature that deals (in rather different ways from ours) with information leakage in two-party protocols. The original paper on zero-knowledge allowed for some information *leakage* to the verifier [GMR89]. This notion was further explored by Goldreich and Petrank [GP99].

Two-party protocols for fair exchange have a problem of early termination by a malicious party. One goal is to design protocols that minimize the *advantage* of the early terminator over the honest party, measured as the difference between the number of bits of recovered messages by each party. Of course, the progress of two-party fair exchange research has primarily focused on increased inefficiency to achieve *less leakage* to the early terminator. This is backwards from our motivation. For examples, refer to [lMR83,Cle89].

Some two-party protocols for computing specific functions allowed some leakage of information [FNW96]. Bar-Yehuda et al. [BYCKO93] consider tradeoffs of information leakage and round complexity for two-party secure computation where the parties are computationally unbounded but non-malicious. Abadi et al. [AFK87] give a model for information leakage to allow a computationally bounded party to compute a function privately, with the help of a computationally unbounded party.

### 1.2 Outline of the Paper

We study *Yao's garbled circuit* protocol in the presence of malicious behavior. We show that the Fairplay scheme of [MNPS04] is still vulnerable to a type of malicious behavior, and suggest a simple way of fixing it. Then, we introduce two different schemes for preventing malicious behavior (1) *Committed-input* scheme, (2) *Equality-checker* scheme. Both constructions have exponentially small error probabilities. We provide and compare communication and computation cost for the designed protocols, both asymptotically and with more concrete measurements. Then, we develop a weaker definition of security for two-party computation that allows for leakage of some information. We then design efficient variations of *Yao's* protocol that are secure in the weaker model. We hope that this weaker definition of security suggests a reasonable tradeoff between efficiency and security, and allows for more efficient and practical implementations of secure two-party protocols.

In Section 2, we review some preliminary concepts. We also give a description of *Fairplay* scheme of [MNPS04]. In Section 3, we mention a vulnerability (against malicious behavior) in Fairplay, and describe our *Committed-input* and *Equality-checker* schemes. In Section 4, we introduce our *k-leaked* model of security, and suggest several efficient constructions that are secure in that model.

## 2 Preliminaries

### Two-Party Computation

A two-party computation is cast by specifying a random process that maps pairs of inputs (one input per each party) to pairs of outputs (one for each party). We refer to such a process as the desired functionality, denoted $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ where $f = (f_1, f_2)$. For every pair of input $x, y \in \{0,1\}^n$, the output-pair is a random variable $(f_1(x,y), f_2(x,y))$ ranging over pairs of strings. The first party wishes to obtain $f_1(x,y)$ and the second party wishes to obtain $f_2(x,y)$.

The security definition for two-party computation varies depending on whether the adversary is *malicious*, or *semi-honest*. A semi-honest (honest-but-curious) adversary follows the steps of the protocol, but does not hesitate to learn more information using the transcripts of messages it receives. On the other hand, a malicious party can behave in an arbitrary way. In this paper, we are concerned with computationally bounded *malicious* adversaries. The definitions we use are according to [Gol04]. These definitions compare the adversaries in the *real-model* with those in an *ideal-model* in which the parties have a *trusted party* at their disposal. Loosely speaking, a two-party protocol is secure if for any admissible pair of parties $(A, B)$ in the *real-model*, there is an admissible pair of parties $(A', B')$ in the *ideal model* where the outputs of the two executions are indistinguishable. A pair is admissible if at least one of the parties in the pair is *honest*. Intuitively, a secure protocol is required to work *correctly*, and to provide *privacy* for the honest participant.

In Section 4, we will present a tweaked version of these definitions that allows a malicious party to learn $k$ bits of extra information.

Oblivious transfer is a special two-party protocol introduced by Rabin [Rab81]. We need the *1-out-of-2* oblivious transfer where $x = (z_0, z_1)$, $y = \sigma$, $f_1(x,y) =$ *empty*, and $f_2(x,y) = z_\sigma$. Several oblivious transfer protocols that are secure in presence of *malicious* or *semi-honest* adversaries exist.

### Yao's garbled circuit protocol

*Yao's garbled circuit* [Yao86] is the first general purpose protocol designed for secure two-party computation. In this protocol, the function being computed is a polynomial size circuit. The first party computes the garbled form of the circuit in the following way:

He assigns two random strings $K_{j,0}$ and $K_{j,1}$ to every wire $j$ in the circuit. These random strings correspond to values 0 and 1, respectively. He then computes a garbled truth table for every gate in the circuit. For this purpose, he uses the random strings as keys to a symmetric encryption scheme, to encrypt the corresponding key for the output wire. He also creates a table that translates the garbled form of the output wires to their actual values (0, or 1). He sends the garbled circuit, and the garbled strings corresponding to his input, to the second party. The second party learns the garbled form of his input bits through a series of oblivious transfers. The second party computes the garbled circuit, gate by gate, and obtains the output in the garbled form. He can then use the translation table to find the actual value of his output. We refer the reader to [LP04] for a complete description of *Yao's protocol* and the proof of its security in the *semi-honest* case.

Yao's protocol, in this form, is not secure when the parties are malicious. Classical ways of making two-party protocols secure against malicious adversaries exist [Gol04] (based on the zero-knowledge compilation technique of [GMW87], and [GMW86]). In particular, the circuit garbler would need to accompany the garbled circuit with a zero-knowledge proof that the circuit is built correctly, and that it computes the desired functionality. Furthermore, the circuit evaluator would need to accompany his final message with a zero-knowledge proof that the output is the result of performing the desired functionality on the inputs exchanged in previous steps of protocol. The general zero-knowledge proofs are quite inefficient and no efficient alternative zero-knowledge proofs are designed for this purpose.

**Fairplay Scheme**

One of the main sources of malicious behavior in Yao's garbled circuit protocol is the ability of the circuit garbler to garble and send a wrong circuit. Malkhi et al. [MNPS04] use a simple cut-and-choose construction which reduces the probability of making the wrong garbled circuit to $\frac{1}{m}$, where $m$ is number of circuits sent by circuit garbler (Bob) to circuit evaluator (Alice).

The following is the Fairplay scheme described in [MNPS04]. We only consider the steps after both parties know the description of the circuit they want to compute.

1. Bob constructs $m$ garbled/encrypted circuits and sends them to Alice. Alice randomly chooses one of the circuits that will be evaluated.
2. Bob exposes the secrets of the other $m - 1$ garbled/encrypted circuits, and Alice verifies them against her reference circuit.
3. Bob specifies his inputs and sends them to Alice in garbled form. Alice inserts Bob's inputs in the garbled/encrypted circuits she chose to evaluate.
4. Alice specifies her inputs, and then Alice and Bob engage in oblivious transfers (OTs) in order for Alice to receive her inputs (in garbled form) from Bob. Bob learns nothing about Alice's inputs.

5. Alice evaluates the chosen garbled/encrypted circuit, finds the garbled outputs of both her and Bob, and sends the relevant outputs to Bob.
6. Each party interprets his or her garbled outputs and prints the result.

In this scheme, the probability that Bob sends the wrong circuit and does not get caught is $\frac{1}{m}$. However, Alice is still vulnerable to a different type of malicious behavior from Bob. We will describe this vulnerability in the following section.

## 3 Preventing Malicious Behavior in Yao's Protocol

**A vulnerability in Fairplay, and how to correct it**

In step 4 of the Fairplay protocol, parties engage in oblivious transfers in order for Alice to get the garbled form of her inputs. In any of the oblivious transfers, Bob can change the order of two random strings corresponding to 0 and 1. Note that changing the order of strings is not a malicious behavior in an oblivious transfer, but becomes a malicious behavior in the above protocol.

To state the vulnerability more formally, note that Bob can *flip* any of Alice's input bits without Alice's detection. Let $x_1, x_2, \ldots, x_n$ be the bit values associated with input wires. Let $W_A$ be the input wires owned by Alice, and let $W_B$ be the input wires owned by Bob: $W_A \cup W_B = [1..n]$ while $W_A \cap W_B = \phi$. For any $S \subseteq [1..n]$, let $flip_S(x_1, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$, where $y_i = 1 - x_i$ for all $i \in S$, and $y_i = x_i$ for all $i \notin S$. Then Bob can fool Alice into computing $f(flip_S(x_1, \ldots, x_n))$ for any $S \subseteq W_A$. It is important to note that this behavior is not allowed in the *ideal model*.

There is a simple solution to this problem. We will require Bob to include a commitment $z_{j,i,b}$ of the tuple $(j, i, b, K_{i,b}^{(j)})$ for every circuit copy $j \in [1..m]$, and every input wire $i \in W_A$, and every input value $b \in \{0, 1\}$, where $K_{i,b}^{(j)}$ denotes the random string corresponding to bit value $b$ of wire $i$ in circuit $j$ (let $w_{j,i,b}$ be the corresponding witnesses for decomittal). The purpose of these commitments is to bind the random key strings with their corresponding bit values (0 or 1). Bob reveals the witnesses for all of the commitments except for those corresponding to the one circuit copy chosen by Alice. Alice can verify that the exposed commitments are correctly computed. For the remaining circuit $s$, Bob will obliviously transfer $(K_{i,0}^{(s)}, w_{s,i,0})$ or $(K_{i,1}^{(s)}, w_{s,i,1})$ for all $i \in W_A$. Alice can use the witnesses to verify that she has received the correct key string. Bob can only cheat with probability $\frac{1}{m}$.

Now, we want to make the cheating probability exponentially small in $m$. The idea mentioned in [Pin03] is that Alice chooses a fraction of circuits randomly. Bob exposes the secrets of those circuits. Alice evaluates the rest of the circuits and accepts the majority output as the correct output of the protocol. For the cut-and-choose protocols to work properly, Bob must be forced to give the same input to *most* of the circuits evaluated by Alice. Pinkas [Pin03] suggests the use of proofs of partial knowledge to achieve this goal, but defers the detail of the actual construction.

Next, we design two schemes for making the cheating probability exponentially small in $m$. In the following two protocols, we assume that only Alice needs to see the output of the protocol. Based on this assumption, both schemes are secure in presence of *malicious* adversaries.

### 3.1 Committed-input Scheme

In this scheme, we will use *proof of equality* of discrete-log commitments [Ped91]. To commit to a value $x$, one generates a random value $r$ and calculates $g^x h^r$, where $g$ is a generator of group $G$ and $h$ is a random element of $G$. The commiter should not know the discrete-log of $h$ base $g$. To prove that $g^x h^{r_1}$ and $g^y h^{r_2}$ are commitments to the same value, the commiter sends $r_2 - r_1$ to the verifier. The verifier can calculate $\frac{g^y h^{r_2}}{g^x h^{r_1}}$ and verify that the result of division is in fact $h^{r_2 - r_1}$. Please refer to [Ped91] for more detail. Note that any commitment scheme with an efficient *proof-of-equality* can be used in our construction. We focus on the Pedersen commitments for simplicity, and to help with our concrete complexity analysis in Section 3.3.

In the *Committed-input* scheme, Bob computes $K_{i,b}^{(j)} = g^b h^{r_{i,b}^{(j)}}$ for random $r_{i,b}^{(j)}$, for every $j \in [1..m]$, for every input wire $i \in W_B$ owned by Bob, and for every input value $b \in \{0, 1\}$.[1] Bob chooses random keys for all of the other wires, including Alice's input wires. He also includes the commitment $z_{j,i,b}$ of the tuple $(j, i, b, K_{i,b}^{(j)})$ for all of Alice's input wire keys (let $w_{j,i,b}$ be the corresponding witness for decommittal). The protocol follows:

1. Bob and Alice agree on a group $G$, and a generator $g$. Alice sends a random element $h \in G$ to Bob.
2. Bob sends the garbled circuits $C^{(j)}$ for every $j \in [1..m]$ (including the translation tables of output wires.) He also sends $(j, i, z_{j,i,0}, z_{j,i,1})$ for every $j \in [1..m]$ and every input wire $i \in W_A$ (commitments in random order).
3. Alice randomly chooses a subset $S \subset [1..m]$, where $|S| = \frac{m}{2}$.
4. Bob exposes all the secrets of circuit $C^{(j)}$ for all $j \in S$. Then, he sends witnesses $r_{i,0}^{(j)}, r_{i,1}^{(j)}$ for every $j \in S$, and $i \in W_B$. He also sends witnesses $w_{j,i,0}, w_{j,i,1}$ for every $j \in S$ and every $i \in W_A$.
5. Alice verifies that all the exposed garbled circuits and commitments were computed correctly. In addition, she verifies that the commitments to 0's were used in the circuit as garbled forms of 0 and commitments to 1 were used as garbled forms of 1.
6. Renumber the remaining garbled circuits as $C^{(1)}, \ldots, C^{(\frac{m}{2})}$. Bob sends to Alice $K_{i,b_i}^{(j)}$ for every $j \in [1..(m/2)]$ and every $i \in W_B$. He also sends $\delta_i^{(j+1)} = r_{i,b_i}^{(j+1)} - r_{i,b_i}^{(j)}$ for every $j \in [1..(m/2) - 1]$ and every $i \in W_B$, where $b_i$ is Bob's input for wire $i$.

[1] If the length of these commitments does not match the length chosen for the random strings, we can use a hash function to map the commitments to strings of the required length.

7. Alice verifies that $K^{(1)}_{i,b_i}, \ldots, K^{(m/2)}_{i,b_i}$ are all commitments to the same value: $K^{(j+1)}_{i,b_i}/K^{(j)}_{i,b_i} = h^{\delta_i^{j+1}}$ for all $j \in [1..(m/2)-1]$. She does so for all $i \in W_B$.
8. Alice specifies her input. Alice and Bob engage in oblivious transfers in order for Alice to receive her input bits in garbled form. Bob uses a single oblivious transfer to give Alice one of the two tuples $(K^{(1)}_{i,0}, w_{1,i,0}, K^{(2)}_{i,0}, w_{2,i,0}, \ldots, K^{(m/2)}_{i,0}, w_{m/2,i,0})$ or $(K^{(1)}_{i,1}, w_{1,i,1}, K^{(2)}_{i,1}, w_{2,i,1}, \ldots, K^{(m/2)}_{i,1}, w_{m/2,i,1})$ depending on whether her value for input wire $i$ is 0 or 1. This Oblivious Transfer is done for every $i \in W_A$.
9. Alice verifies that these received input wire values and witnesses are consistent. Then, Alice executes all $m/2$ garbled circuits and outputs the majority value of the outputs.

**Proof of Security**

We assume the basic building blocks for Yao's garbled circuit protocol: secure 1-out-of-2 oblivious transfer and secure symmetric encryption. In addition, we assume the security of Pedersen commitments (discrete log assumption).

**Lemma 1.** *With probability more than $1 - (\frac{1}{2})^{\frac{m}{4}}$, the majority of evaluated circuits are correct and have the same input, or Bob will get caught.*

*Proof of lemma:* The probability that more than half of the remaining $\frac{m}{2}$ circuits were wrong, and were not detected by Alice, is less than $\binom{3m/4}{m/2} \,/\, \binom{m}{m/2} < (\frac{1}{2})^{\frac{m}{4}}$. Therefore, the majority of circuits are correct with high probability, which means that the disc-log commitments corresponding to those circuits are also correct. Hence, Bob has to give the same input for those circuits or he will get caught during the verification (step 7).

The following two claims complete the security argument. Due to lack of space, proofs of the following two claims are not included in this extended abstract.

**Claim 1** *The* Committed-input *scheme is secure when Bob (circuit garbler) is malicious (inverse exponential probability of undetected cheating).*

**Claim 2** *The* Committed-input *scheme is secure when Alice (circuit evaluator) is malicious.*

### 3.2 Equality-checker Scheme

In this scheme, we will avoid any *exponentiation* other than the ones computed for OTs. Before describing the scheme, lets define the equality-checkers used in the scheme. Let $z_{j,j',i,b}$ be Bob's commitment to the tuple $(j, j', K^{(j)}_{i,b}, K^{(j')}_{i,b})$ and let $w_{j,j',i,b}$ be the corresponding witness for decommittal. Bob computes these commitments for every $j, j'$ such that $1 \le j < j' \le m$, for every $i \in W_B$, and for every $b \in \{0, 1\}$. The idea is that a correctly built commitment binds the two random key strings that correspond to the same bit value for the same input wire, but in two different circuits. Alice can verify that Bob's input to two circuits $C^{(j)}$ and $C^{(j')}$ are equal if she is given the witnesses to the commitments $z_{j,j',i,b_i}$ for every $i \in W_B$, where $b_i$ is Bob's input bit for wire $i$. An *equality-checker* between circuits $C^{(j)}$ and $C^{(j')}$ is the collection of $z_{j,j',i,b}$ for all $i \in W_B$ and $b \in \{0,1\}$. Working with equality-checkers instead of individual commitments makes the proofs simpler.

1. Bob constructs $m$ garbled/encrypted circuits and sends them to Alice. He also sends the $z_{j,i,b}$ commitments and the $m(m-1)/2$ equality-checkers described above.
2. Alice randomly chooses a subset $S \subset [1..m]$, where $|S| = \frac{m}{2}$ and sends $S$ to Bob.
3. Bob exposes the secrets of circuits $C^{(j)}$ for all $j \in S$. Then, he sends witnesses $w_{j,i,b}$ for all $j \in S$, all $i \in W_A$, and all $b \in \{0,1\}$. He also sends witnesses $w_{j,j',i,b}$ for all $j, j' \in S$, all $i \in W_B$, and all $b \in \{0,1\}$ (these are the $(\frac{m}{2})(\frac{m}{2}-1)/2$ equality-checkers corresponding to pairs of revealed circuits). Alice verifies that the garbled circuits and commitments were computed correctly.
4. Renumber the remaining garbled circuits as $C^{(1)}, \ldots, C^{(m/2)}$. Bob sends the keys $K^{(j)}_{i,b_i}$ for every $j \in [1..\frac{m}{2}]$ and $i \in W_B$. He also sends witnesses $w_{j,j',i,b_i}$ for every $1 \le j < j' \le \frac{m}{2}$, and every $i \in W_B$, where $b_i$ is his input for wire $i$.
5. Alice uses the witnesses $w_{j,j',i,b_i}$ to verify that Bob's input to all the circuits are the same.
6. Alice and Bob engage in oblivious transfers in order for Alice to receive her input bits in garbled form. Bob uses a single oblivious transfer to give Alice one of the two tuples $(K^{(1)}_{i,0}, K^{(2)}_{i,0}, \ldots, K^{(m/2)}_{i,0})$ or $(K^{(1)}_{i,1}, K^{(2)}_{i,1}, \ldots, K^{(m/2)}_{i,1})$ (depending on whether her value for input wire $i$ is 0 or 1). This Oblivious Transfer is done for every $i \in W_A$.
7. Alice will evaluate the $\frac{m}{2}$ garbled circuits and print the majority output as the correct output.

**Proof of Security**

We assume the basic building blocks for Yao's garbled circuit protocol: secure 1-out-of-2 oblivious transfer and secure symmetric encryption. We also assume the security of the commitment scheme (which can be built from one way functions).

**Lemma 2.** *With probability more than $1-(1/2)^{\frac{m}{6}}$, more than $\frac{2}{3}$ of the $\frac{m}{2}$ circuits are correct, or Bob will get caught.*

*Proof of Lemma:* (by contradiction) Let's assume that at most $\frac{2}{3}$ of the $\frac{m}{2}$ circuits are correct. This means that at least $\frac{m}{6}$ of the circuits are wrong. The probability that Alice doesn't detect those $\frac{m}{6}$ is less than $\binom{5m/6}{m/2} / \binom{m}{m/2} < (\frac{1}{2})^{\frac{m}{6}}$.

**Lemma 3.** *With probability more than $1-(1/2)^{\frac{m}{6}}$, at least $\frac{5}{6}$ of Bob's $\frac{m}{2}$ inputs are the same, or Bob will get caught.*

Proof: See Appendix for the proof.

**Claim 3** *The* Equality-checker *scheme is secure when Bob (circuit garbler) is malicious (inverse exponential probability of undetected cheating).*

*Proof sketch:* Now we know that Lemmas 2 and 3 are correct. In other words, at least $\frac{5}{6}$ of inputs are the same, and more than $\frac{2}{3}$ of circuits are correct with high probability. This implies that more than $\frac{2}{3} - \frac{1}{6} = \frac{1}{2}$ of the circuits are correct and have the same inputs, and hence, the majority output is the correct output. according to the union-bound this will happen with probability greater than $1 - (1/2)^{\frac{m}{6}} + 1 - (1/2)^{\frac{m}{6}} - 1 = 1 - 2(1/2)^{\frac{m}{6}}$.

Consider a strategy $B$ for Bob in the real model. If Alice aborts the protocol, we are done (Bob is caught and he doesn't learn anything). But if Alice doesn't abort, she will respond with the majority output $O$ which is equal to $f(x_a, x_{maj})$ with high probability. Here, $x_a$ is Alice's input to the circuit, and $x_{maj}$ is Bob's input to majority of the circuits. Bob's view of the protocol includes the OTs, and output $O$. Since he doesn't learn anything about Alice's input during the OTs, he can simulate them on his own using a simulator $S_1$.

The adversary $B^{'}$ in the ideal model will send the input $x_{maj}$ to the *trusted-party* and get back $f(x_a, x_{maj})$ as the output. He can use the simulator $S_1$ to simulate the OTs, and emulate $B$'s strategy step by step, and the view of the protocol will be indistinguishable.

The following claim completes the security argument (Proof is omitted due to lack of space).

**Claim 4** *The* Equality-checker *scheme is secure when Alice (circuit evaluator) is malicious.*

### 3.3 Communication and Computation Analysis

To measure the communication and computation complexity of the schemes, we introduce the parameters $m$, $I$, $O$ and $g$. $m$ is number of garbled/encrypted circuits sent to Alice. $I$ and $O$ are the number of input and output bits respectively (Bob and Alice combined). $g$ denotes the number of gates in the circuit. Whenever we want to consider one party's input or output, we will use the proper subscript.

#### Asymptotic Analysis

We will measure the communication and computation cost where the goal is to achieve an error probability as small as $\epsilon$. To measure the computation cost, we split the computation into two types of operations: (1) exponentiations, and (2) everything else. In our protocols, OTs and disc-log commitments are from the first category, while the symmetric encryptions and other commitments used in the protocol are in the second category.

In the Fairplay scheme (with the vulnerability fixed as suggested), to achieve the required error probability, we need $m = \frac{1}{\epsilon}$ circuits. But for the Committed-input scheme and Equality-checker scheme, $m = O(ln(\frac{1}{\epsilon}))$. We summarize the communication and computation complexities in table 1. Note that $t = O(ln(\frac{1}{\epsilon}))$ is the security parameter (for a successful cheating probability of $\epsilon$).

| Scheme | Symmetric Enc. | Exponentiations | Communication Complexity |
|---|---|---|---|
| Fairplay | $O(\frac{1}{\epsilon}g)$ | $O(I)$ | $O(2^t g)$ |
| Committed-input | $O(ln(\frac{1}{\epsilon})g)$ | $O(ln(\frac{1}{\epsilon})I)$ | $O(tg)$ |
| Equality-checker | $O(ln(\frac{1}{\epsilon})g + ln(\frac{1}{\epsilon})^2 I)$ | $O(I)$ | $O(tg + t^2 I)$ |

**Table 1.** Computation and Communication complexities

**More Concrete Analysis**

We will try to measure the computational cost of all three constructions more precisely. We take into account all the encryptions, commitments and exponentiations, and include even the constant factors. We will measure the computational cost for 4 different circuits (AND[2], Billionaires[3], PIR[4], Median[5]). We borrow the circuits and their sizes from [MNPS04].

*Fairplay*: In the Fairplay scheme, to achieve an error probability $\epsilon$, the total number of cryptographic operations are: $\frac{1}{\epsilon}(4g+2O+2I_A)$ symmetric encryptions and $2I_A$ exponentiations.

*Committed-input*: To achieve an error probability of $\epsilon$, the total number of cryptographic operations are: $\frac{4}{ln(2)}ln(\frac{1}{\epsilon})(4g + 2O + 2I_A)$ symmetric encryptions and $2I_A + \frac{8}{ln(2)}ln(\frac{1}{\epsilon})I_B$ exponentiations.

*Equality-checker*: We have $\frac{6}{ln(2)}\ln(\frac{1}{\epsilon})(4g+2O+2I_A) + \frac{72}{ln(2)^2}(ln(\frac{1}{\epsilon}))^2 I_B$ symmetric encryptions, and $2I_A$ exponentiations.

Tables 2, 3 and 4 give the computational costs for the four mentioned circuits, for four different error probabilities. Each entry includes two integers, representing the number of symmetric encryptions and exponentiations, respectively.

Since *Equality-checker* and *Fairplay* have the same number of exponentiations, it is easy to compare their computational cost. You can see that for all four circuits, if we require an error probability of $\frac{1}{1000}$ or smaller, the computational cost of *Equality-checker* scheme is lower. However, if an error probability of $\frac{1}{100}$ or larger is enough, Fairplay is a better choice. Therefore, the choice of efficient construction seems to depend on the likelihood of malicious behavior

[2] Performs bit-wise AND on two inputs of size 8. The circuit has 32 gates.
[3] Compares two 32-bit integers. The circuit has 256 gates.
[4] Bob's input size is 480 bits and Alice's input size is 6 bit. The circuit has 1229 gates.
[5] finds the median of two sorted arrays. The input for both Alice and Bob are ten 16-bit numbers. The circuit size is 4383 gates.

| Error probability | And | Billionaires | PIR | Median |
|---|---|---|---|---|
| $\frac{1}{100}$ | $(176 * 10^2, 16)$ | $(1092 * 10^2, 64)$ | $(4976 * 10^2, 12)$ | $(17916 * 10^2, 320)$ |
| $\frac{1}{1000}$ | $(176 * 10^3, 16)$ | $(1092 * 10^3, 64)$ | $(4976 * 10^3, 12)$ | $(17916 * 10^3, 320)$ |
| $\frac{1}{10000}$ | $(176 * 10^4, 16)$ | $(1092 * 10^4, 64)$ | $(4976 * 10^4, 12)$ | $(17916 * 10^4, 320)$ |
| $\frac{1}{1000000}$ | $(176 * 10^6, 16)$ | $(1092 * 10^6, 64)$ | $(4976 * 10^6, 12)$ | $(17916 * 10^6, 320)$ |

**Table 2.** computational cost for Fariplay scheme

| Error probability | And | Billionaires | PIR | Median |
|---|---|---|---|---|
| $\frac{1}{100}$ | $(4677, 441)$ | $(29020, 1764)$ | $(132239, 25524)$ | $(476125, 8824)$ |
| $\frac{1}{1000}$ | $(7015, 653)$ | $(43530, 2615)$ | $(198358, 38280)$ | $(714187, 13076)$ |
| $\frac{1}{10000}$ | $(9354, 866)$ | $(58040, 3465)$ | $(264478, 51036)$ | $(952250, 17328)$ |
| $\frac{1}{1000000}$ | $(14031, 1291)$ | $(87061, 5166)$ | $(396717, 76549)$ | $(1428375, 25832)$ |

**Table 3.** Computational cost for Committed-input scheme

and the magnitude of damage it can have in the environment the protocol is being employed.

David Woodruff [Woo06] has proposed a modification to the Equality-checker scheme using expander graphs. Bob associates his $m$ circuits with the vertices of an expander graph, and then commits only to those pairs of circuits that correspond to edges of this graph. There are explicit constructions of expander graphs for which this saves a factor of $\Theta(m)$ in the communication complexity, while preserving the security properties of the protocol. This is a nice asymptotic improvement, although it is unclear what the savings would be for the small values of $m$ that might be used in practice.

## 4 How to Leak Information

The idea explored in this section is to weaken the notion of security by allowing a *malicious* party to learn $k$ bits of information about the other party's input, in addition to the output of protocol. A *semi-honest* party should still only learn the output of the protocol.

Loosely speaking, a two-party protocol $\pi$ between two parties $A$ and $B$ for computing $f(x_a, x_b)$, leaks only $k$ bits of information if all the malicious party $A$ (symmetrically, $B$) can learn from protocol $\pi$, it can also learn given the output $f(x_a, x_b)$ and an additional value $g(x_b)$ for a $g$ of her choice in $G$, where $G$ is a family of functions and $G \subseteq \{g | g : \{0,1\}^* \rightarrow \{0,1\}^k\}$.

The definition of security of a two-party protocol (refer to [Gol04]), compares the execution of admissible adversaries in the *real-model* and *ideal-model*. In order to formally incorporate the leakage of information in our definition of security, we need to change the definition of *ideal-model*. We call this new model the *k-leaked model*.

| Error probability | And | Billionaires | PIR | Median |
|---|---|---|---|---|
| $\frac{1}{100}$ | $(19700, 16)$ | $(94380, 64)$ | $(961112, 12)$ | $(968439, 320)$ |
| $\frac{1}{1000}$ | $(39100, 16)$ | $(179708, 64)$ | $(2013733, 12)$ | $(1643347, 320)$ |
| $\frac{1}{10000}$ | $(64882, 16)$ | $(290462, 64)$ | $(3447731, 12)$ | $(2445380, 320)$ |
| $\frac{1}{1000000}$ | $(135460, 16)$ | $(588243, 64)$ | $(7459858, 12)$ | $(4430824, 320)$ |

**Table 4.** Computational cost for Equality-checker scheme

**k-leaked model**

In this model of computation, parties have a *semi-trusted* party at their disposal. Execution in the *k-leaked model* proceeds as follows:

- ***Inputs:*** Each party obtains an input denoted $u$.
- ***Sending inputs to the semi-trusted party:*** An honest party always sends $u$ to the semi-trusted party. A malicious party may, depending on $u$ (as well as an auxiliary input and its coin tosses), either abort or send some other $u^{'} \in \{0,1\}^{|u|}$ to the semi-trusted party.
- ***Malicious party asks for $k$ bits of information:*** In case either party has aborted, the *semi-trusted party* replies to both parties with a special symbol, denoted $\perp$. Otherwise, the *semi-trusted party* has an input pair $(x, y)$. A malicious party can choose a function $g \in G$, where $G \subseteq \{g | g : \{0,1\}^* \rightarrow \{0,1\}^k\}$, and ask the *semi-trusted party* for the value of $g$ at the other party's input. The *semi-trusted party* answers accordingly.
- ***The semi-trusted party answers the first party:*** The semi-trusted party answers the first party with $f_1(x, y)$.
- ***The semi-trusted party answers the second party:*** In case the first party is malicious, it may, depending on its input and the answer it received, decide to *stop* the semi-trusted party. In this case the semi-trusted party sends $\perp$ to the second party. Otherwise (i.e. if not stopped), the semi-trusted party sends $f_2(x, y)$ to the second party.
- ***Outputs:*** An honest party always outputs the message it has obtained from the semi-trusted party. A malicious party may output an arbitrary function of its initial input and the message it has obtained from the semi-trusted party.

Note that we borrowed the definition of *ideal-model* from [Gol04] and made the necessary adjustments to obtain a definition for *k-leaked model*. Now, we can easily obtain a definition for our weakened notion of security by replacing the *ideal-model* by the *k-leaked model* in the security definition of [Gol04].

Loosely speaking, for any admissible pair of parties $(A, B)$ in the *real-model*, there is an admissible pair of parties $(A^{'}, B^{'})$ in the *k-leaked model* where the outputs of the two executions are indistinguishable. A pair is admissible if at least one of the parties in the pair is *honest*. We will take advantage of this fact when designing our protocols.

### 4.1 How to use the new definition

In this section, we will describe a method for making *Yao's garbled-circuit* protocol secure against malicious behavior in the *1-leaked model*, where $G = \{g | g : \{0,1\}^* \rightarrow \{0,1\}\}$. Note that we have already designed two protocols for making *Yao's protocol* secure against malicious behavior. Our new construction is interesting because it is simple, generic, and more efficient. Particularly, it has the same communication and computation complexity as *Yao's garbled-circuit* protocol for semi-honest parties (Proof of security in the *1-leaked* model is omitted from this extended abstract).

#### The protocol

The protocol takes place between Alice and Bob who want to compute $f(x_a, x_b)$ where $x_a$ is Alice's input and $x_b$ is Bob's.[6]

1. Alice creates a garbled circuit for computing $f$. She sends the garbled circuit, her garbled input, and a translation table for output wires to Bob.
2. Alice and Bob engage in a series of oblivious transfer protocols so that Bob learns the garbled form of his inputs.
3. Bob computes the circuit and translates the output strings to their actual value using the translation table. Lets call this output $O_1$.
4. Bob creates a garbled circuit for computing $f$. He sends the garbled circuit, his garbled input, and a translation table for output wires to Alice.
5. Alice and Bob engage in a series of oblivious transfer protocols so that Alice learns the garbled form of her inputs.
6. Alice computes the circuit and translates the output strings to their actual value, using the translation table. Lets call this value $O_2$.
7. Alice and Bob engage in a secure protocol (against malicious behavior) that returns 1 if $O_1 = O_2$ and 0 otherwise (This is where a malicious party can learn one extra bit). Bob and Alice need to prove to each other that they actually use $O_1$ and $O_2$ as their input to this sub-protocol.
8. If the answer is 0, parties output $\perp$, and abort.
9. Bob and Alice output $O_1$ and $O_2$ respectively.

#### Instantiating step 7 of the protocol

The sub-protocol in step 7 needs to return 0 if the inputs are not the same. In addition, any party using an input different from $O_1$ or $O_2$ should get caught. It is in fact easy to achieve the latter by requiring the parties to incorporate in the sub-protocol, the garbled form of the output they received. Note that if the translation tables are carefully constructed, a party computing a garbled

[6] Please note that the following protocol only considers the case where both parties share the same output. This doesn't effect the generality of the protocol since any two-party computation in which parties have different outputs can be solved using protocols in which both parties share the same output (please refer to [LP04]).

circuit can only learn the garbled strings corresponding to its own output bits and not the complements. Hence, it is easy for the other party, who created the garbled circuit in first place, to verify the correctness of it. We need a conditional disclosure protocol as described in [AIR01].

In the following protocol, $O_1$ is the output received by Bob (in binary) and $W_b = w_1||w_2\dots||w_n$ is the garbled form of $O_1$. Furthermore, the output received by Alice is $O_2$ with the garbled form $W_a = w_1||w_2\dots||w_n$. Then, the protocol, at a high level, is as follows:

1. Alice discloses $W_a$ to Bob if $O_1 = O_2$, and a random value $r_a \in \{0,1\}^{|W_a|}$ otherwise.
2. Bob calculates his own version of $W_a$ using $O_1$ and the garbled strings corresponding to the output wires (Bob created the garbled circuit). He verifies that his calculated version is equal to the value he received from Alice. If not, he aborts.
3. Bob discloses $W_b$ to Alice if $O_1 = O_2$, and a random value $r_b \in \{0,1\}^{|W_b|}$ otherwise.
4. Alice verifies the equality in a way similar to (step 2).

**Computational-cost** Note that by leaking one extra bit, we made the protocol much more efficient. We decreased the communication cost to only twice that of *Yao's garbled circuit* for semi-honest parties. The same is true regarding the cost of computation. Below is a measure of how the protocol performs, both asymptotically and in more concrete terms. As before, for the concrete measurements, the first component shows number of symmetric encryptions while the second component counts number of exponentiations performed. Comparing this table with Tables 2, 3,and 4 shows the dramatic improvement in the computation cost.

| symmetric Enc. | exponentiations | And | Billionaires | PIR | Median |
|---|---|---|---|---|---|
| $O(g)$ | $O(I)$ | $(352, 34)$ | $(2184, 130)$ | $(9952, 26)$ | $(35832, 642)$ |

**Table 5.** Computational-cost of the scheme

### Non-interactive Computations

The protocol we used above to make *Yao's garbled-circuit* protocol secure against malicious behavior is generic enough that it can be used in different contexts as well. Particularly, any *non-interactive*[7] two-party protocol which is secure

[7] We call a protocol non-interactive if one party sends his input (in some form) to the second party. Then, the second party computes the functionality on his own (no interaction here).

against semi-honest adversaries can use our scheme to make the protocol secure against malicious behavior in the *1-leaked model*. More generally, any non-interactive two-party protocol that is secure in the *semi-honest* version of the *k-leaked model*, can be made secure in the malicious version of *(k+1)-leaked model*. It is important to note that step 7 of the protocol should be instantiated appropriately.


## Acknowledgements

We would like to thank David Woodruff and the anonymous reviewers for their helpful suggestions.


## References


[AFK87] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 195–203. ACM Press, 1987.

[AIR01] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. *Eurocrypt*, 2001.

[BYCKO93] R. Bar-Yehuda, B. Chor, E. Kushilevitz, and A. Orlitsky. Privacy, additional information, and communication. *IEEE Transactions on Information Theory*, 1993.

[Cle89] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 573–588. Springer-Verlag, 1989.

[FNW96] R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Commun. ACM*, 39(5):77–85, 1996.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proofs. *Proceedings of of the 27th FOCS, pages 174-187*, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. *In proceedings of 19th Annual ACM Symposium on Theory of Computing, pages 218-229*, 1987.

[Gol04] O. Goldreich. Foundations of cryptography - volume 2, ch. 7. 2004.

[GP99] O. Goldreich and E. Petrank. Quantifying knowledge complexity. *Computational Complexity*, 8:50–98, 1999.

[lMR83] M. luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. *FOCS*, 1983.

[LP04] Y. Lindell and B. Pinkas. A proof of yao's protocol for secure two-party computation. *eprint archive*, 2004.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay– a secure two-party computation system. *Proceedings of Usenix security*, 2004.

[Ped91] T. P. Pederson. Non-interactive and information-theoritic secure verifiable secret-sharing. 1991.

[Pin03] Benny Pinkas. Fair secure two-party computation. *Eurocrypt, LNCS 2656, Springer-Verlag, pp. 87-105*, 2003.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. *Technical Report Tech., Memo. TR-81, Aiken Computation Labratory, Harvard University*, 1981.

[Woo06] D. Woodruff. unpublished manuscript, 2006.

[Yao86] A. C. Yao. How to generate and exchange secrets. *In Proceedings of the 27th IEEE symposioum on Foundations of Computer science,pages 162-167*, 1986.

# A Proof of Lemma 3

Bob sends $\frac{m}{2}$ garbled inputs for the $\frac{m}{2}$ circuits that Alice will evaluate. Lets denote the set of these inputs $I_B$, where $|I_B| = \frac{m}{2}$. Let $L$ be the largest subset of $I_B$ with equal inputs, where $|L| = k$. In other words, all other subsets of equal inputs have sizes smaller or equal to $k$. We want to prove that, with high probability, $k$ is greater than $\frac{5}{6}.\frac{m}{2} = \frac{5m}{12}$.

Note that Alice has $(\frac{m}{2})(\frac{m}{2} - 1)/2$ equality-checkers that compare the $\frac{m}{2}$ inputs with each other. Some of these equality-checkers might be wrong (malicious Bob), and therefore verify the equality of two inputs that are not equal. We call two equality-checkers *distinct* if they compare the inputs to four different circuits. The next two Claims imply Lemma 3:

*Claim A:* If $k \leq \frac{5m}{12}$, at least $\frac{m}{12}$ of the distinct equality-checkers used by Alice for verification were wrong.

*Claim B:* If Bob sends $\frac{m}{12}$ wrong distinct equality-checkers to Alice, the probability that they are not detected by Alice, and are used by her to verify that Bob's inputs are the same, is less than $(1/2)^{\frac{m}{6}}$.

*Proof of Claim A:* We know that $|L| = k \leq \frac{5m}{12}$. We consider the following two case:

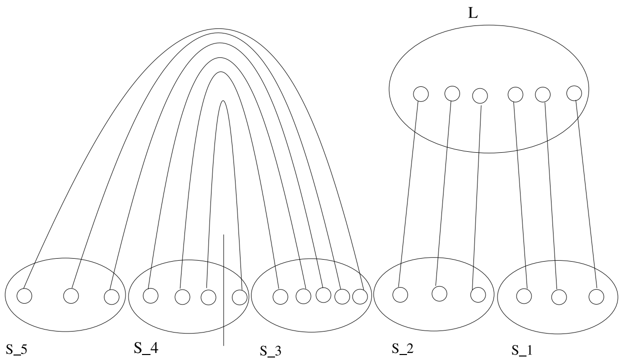


**Fig. 1.** Nodes represent inputs and edges represent equality-checkers

*Case 1:* ($\frac{m}{6} \leq k \leq \frac{5m}{12}$) This implies that $min(|L|, |I_B - L|) = min(k, \frac{m}{2} - k) \geq \frac{m}{2} - \frac{5m}{12} = \frac{m}{12}$. Therefore, there are at least $\frac{m}{12}$ distinct equality-checkers among those that compare the inputs in $(I_B - L)$ with those in $L$. These $\frac{m}{12}$ equality-checkers must have been wrong not to detect that the compared inputs are not the same.

*Case 2:* ($k \leq \frac{m}{6}$) Consider the partition $(S_1, S_2, ..., S_l, L)$ of the set $I_B$ where each subset $S_i$ and $L$ only contain equal inputs (Fig 1.).Note that $S_1 \cup S_2 \ldots \cup S_l \cup L = I_B$, and all the subsets are pairwise disjoint. Lets denote $|S_i| = k_i$ for all $1 \leq i \leq l$. The fact that $k_1 + ... + k_l + k = \frac{m}{2}$, and $(k \leq \frac{m}{6})$ implies that $k_1 + k_2 + ... + k_l \geq k$. Hence, there are $k$ distinct equality-checkers that compare the $k$ inputs in $L$ with inputs in $S_1, S_2, ..., S_j$ for some $1 \leq j < l$ (Fig 1.). This might only cover portions of set $S_j$. These $k$ equality-checkers must have been wrong not to detect that their two inputs weren't equal. But there is more.

We insert the remaining portion of $S_j$ and all of $S_{j+1}, ..., S_l$ in a list (in the same order). We cut the list in half, and pair up each input on the right side of the cut with its counterpart on the left (Fig. 1). In worst case, the cut is in the middle of a subset $S_t$ $(j+1 \leq t \leq l)$. This means that at most $\frac{k_t}{2} < \frac{k}{2}$ of these pairs might include equal inputs (in the same subset). The equality-checkers corresponding to the rest of the pairs compare unequal inputs and must have been wrong not to detect the inequalities. Therefore, there is an additional $(\frac{m}{2} - k - k - k_t)/2 > (\frac{m}{2} - 3k)/2$ wrong equality-checkers. This makes the total number of wrong distinct equality-checkers at least $k + (\frac{m}{2} - 3k)/2 = \frac{m}{4} - \frac{k}{2} > \frac{m}{12}$.

Based on Case 1 and Case 2, if less than $\frac{5}{6}$ of the inputs are the same, and all the equality-checkers confirm the equality of inputs, there are at least $\frac{m}{12}$ wrong distinct equality-checkers among them.

Proof of Claim B: Lets assume that $\frac{m}{12}$ distinct equality-checkers are wrong and are used by Alice for verification. The probability that no two endpoints of any of the $m/12$ equality-checkers was exposed in the previous step is less than $(\frac{1}{2})^{\frac{m}{6}}$.