Efficient Two Party and Multi Party Computation against Covert Adversaries

Vipul Goyal[1], Payman Mohassel[2], and Adam Smith[3]

[1] Department of Computer Science, UCLA
vipul@cs.ucla.edu
[2] Department of Computer Science, UC Davis
pmohassel@ucdavis.edu
[3] Department of Computer Science, PSU
asmith@cse.psu.edu

Abstract. Recently, Aumann and Lindell introduced a new realistic security model for secure computation, namely, security against *covert adversaries*. The main motivation was to obtain secure computation protocols which are efficient enough to be usable in practice. Aumann and Lindell presented an efficient two party computation protocol secure against covert adversaries. They were able to utilize cut and choose techniques rather than relying on expensive zero knowledge proofs.

In this paper, we design an efficient multi-party computation protocol in the covert adversary model which remains secure even if a majority of the parties are dishonest. We also substantially improve the two-party protocol of Aumann and Lindell. Our protocols avoid general NP-reductions and only make a *black box* use of efficiently implementable cryptographic primitives. Our two-party protocol is constant-round while the multi-party one requires a logarithmic (in number of parties) number of rounds of interaction between the parties. Our protocols are secure as per the standard *simulation-based* definitions of security.

Although our main focus is on designing efficient protocols in the covert adversary model, the techniques used in our two party case directly generalize to improve the efficiency of two party computation protocols secure against standard malicious adversaries.

1 Introduction

Secure multi-party computation (MPC) allows a set of n parties to compute a joint function of their inputs while keeping their inputs private. General secure MPC has been an early success of modern cryptography through works such as [Yao86,GMW87,BOGW88,CCD88]. The early MPC protocols used very generic techniques and were inefficient. Hence, now that most of the questions regarding the *feasibility* of secure computation have been addressed (at least in the stand alone setting), many of the recent works have focused on improving the efficiency of these protocols.

The most hostile situation where one could hope to do secure computation is when we have a *dishonest majority*. That is, where up to $(n-1)$ parties could be corrupted and could deviate arbitrarily from the protocol. The feasibility of secure computation in this setting was shown by [GMW87]. Several later results focused on improving its efficiency (often quantified as round complexity).

Most of these constructions use *general zero-knowledge proofs* to compile honest-but-curious MPC protocols into fully malicious MPC protocols. These zero-knowledge compilers are of great theoretical importance but lead to rather inefficient constructions. These compilers make a *non-black-box* use of the underlying cryptographic primitives. To illustrate this inefficiency, consider the following example taken from [IKLP06]. Suppose that due to major advances in cryptanalytic techniques, all basic cryptographic primitives require a full second of computation on a fast CPU. *Non-black-box* constructions require parties to prove in zero-knowledge, statements that involve the computation of the underlying primitives, say a trapdoor permutation. These zero-knowledge protocols, in turn, invoke cryptographic primitives for every gate of a circuit computing a trapdoor permutation. Since (by our assumption) a trapdoor permutation takes one second to compute, its circuit implementation contains trillions of gates, thereby requiring the protocol trillions of second to run. A black box construction, on the other hand, would make the number of invocations of the primitive independent of the complexity of implementing the primitive.

Due to lack of efficient and practical constructions for the case of dishonest majority, a natural question that arises is *"Can we relax the model (while still keeping it meaningful) in a way which allows us to obtain efficient protocols likely to be useful in practice?"*.

One such model is the well known honest majority model. The model additionally allows for the construction of protocols with guaranteed output delivery. Positive steps to achieve efficient protocols in this model were taken by Damgard and Ishai [DI05]. They presented an efficient protocol which makes a black box use of only a pseudorandom generator.

Another such model is the model of *covert adversaries* (incomparable to the model of honest majority) recently introduced by Aumann and Lindell[AL07] (see also [CO99]). A covert adversary may deviate from steps of the protocol in an attempt to cheat, but such deviations are detected by honest parties with good probability (although not with negligibly close to 1). As Aumann and Lindell argue, covert adversaries model many real-world settings where adversaries are willing to actively cheat (and therefore are not semi-honest) but only if they are not caught doing so. This is the case for many business, financial, political and diplomatic settings where honest behavior cannot be assumed but where companies, institutions, or individuals cannot afford the embarrassment, loss of reputation and negative press associated with being caught cheating. They further proceed to design an efficient two-party computation protocol secure against covert adversaries with only blackbox access to the underlying primitives. Their construction applies cut-and-choose techniques to Yao's garbled circuit, and takes advantage of an efficient oblivious transfer protocol secure against covert adversaries. Currently, there is no such counterpart for the case of ≥ 3 parties with dishonest majority.

Our Results:

Multi-party Computation against Covert Adversaries. We construct a protocol for multi-party computation in the covert adversary model. Our protocol provides standard simulation based security guarantee if any number of the parties collude maliciously. Our techniques rely on efficient cut and choose techniques and avoid expensive zero-knowledge proofs to move from honest-but-curious to malicious security. We only make a *black-box use* of *efficiently implementable* cryptographic primitives.

The protocol requires $O(n^3ts|C|)$ bits of communication (and similar computation time) to securely evaluate a circuit C with deterrence $1-\frac{1}{t}$. Here $\frac{1}{t}$ is the noticeable, but small probability with which the cheating parties may escape detection, and s is a cryptographic security parameter. In contrast, the most efficient previously known protocols, due to Katz, Ostrovsky and Smith [KOS03] and Pass [Pas04], require zero-knowledge proofs *about* circuits of size $O(n^3s|C|)$.

The protocol in this paper requires $O(\log n)$ rounds of interaction, due to an initial coin-flipping phase that follows the Chor-Rabin scheduling paradigm [CR87]. The round complexity can be reduced to a constant using non-black-box simulation techniques [Bar02,KOS03,Pas04], but the corresponding increase in computational complexity makes it unlikely that the resulting protocol would be practical.

We remark that there have been a number of two-parties protocols designed using cut and choose techniques [MNPS04,MF06,Woo07,LP07], where one party prepares several garbled circuits while the other party randomly checks a subset of them. However, this paper is the first work to employ such techniques for the design of efficient protocols in the multi-party setting.

Two-party Computation against Covert Adversaries. In a protocol secure against covert adversaries, any attempts to cheat by an adversary is detected by honest parties with probability at least ϵ, where ϵ is the deterrence probability. Therefore, a *high deterrence probability* is crucial in making the model of covert adversaries a practical/realistic model for real-world applications. In this paper we design a *two-party protocol secure against covert adversaries* in which the deterrence probability $\epsilon = 1 - 1/t$, for any value of t polynomial in the security parameter, comes almost for *free* in terms of the *communication complexity* of the protocol. The following table compares our result against that of previous work, where $|C|$ is the circuit size, m is the input size, and s is the statistical security parameter.

Protocol	Communication Complexity
[AL07]	$O(t\|C\| + tsm)$
This paper (section 3.1)	$O(\|C\| + sm + t)$

Two-party Computation against Fully Malicious Adversaries. Although we mainly focus on covert adversaries, we also show how our techniques lead to secure two-party computation schemes against *fully malicious* adversaries. Particularly, by applying our techniques to the existing cut-and-choose protocols, i.e.

[LP07,Woo07,MF06], we improve the communication cost of these protocols without affecting their security guarantees. In this case, our improvement in the communication cost of these protocols is not asymptotic but rather in concrete terms.

Related Work. Katz *et al.* [KOS03] and Pass [Pas04] give the most round-efficient secure MPC protocols with dishonest majority. Ishai et al. [IKLP06], give the first construction for dishonest majority with only black-box access to a trapdoor permutation. Although theoretically very interesting, these approaches are not attractive in terms of efficiency due to the usage of very generic complexity theoretic techniques.

The compiler of Lindell [Lin01] may be applied to achieve constant-round protocols for secure two-party computation. More recent works on secure two-party computation avoid the zero-knowledge machinery (using cut-and-choose techniques), and design efficient protocols with only black-box access to the underlying primitives. Application of cut-and-choose techniques to Yao's garbled circuit was first suggested by Pinkas [Pin03], and further refined and extended in [MNPS04,MF06,Woo07,LP07]. The protocols of [MF06] and [LP07] lead to $O(s|C|+s^2m)$ communication between the parties, while the protocol of [Woo07] only requires $O(s|C|)$ communication where s is the security parameter. Our improvement in the communication cost of these protocols is not asymptotic but rather in concrete terms. Lindell and Pinkas [LP07] also showed how the cut-and-choose techniques could be modified to also yield simulation-based proofs of security. Their ideas can also be applied to [MF06,Woo07]. A different approach for defending against malicious adversaries in two party computation is taken by Jarecki and Shmatikov [JS07]. The basic idea in their work is to have the first party generate a garbled circuit and prove its correctness by giving an efficient number-theoretic zero-knowledge proof of correctness for every gate in the circuit. This protocol is more communication efficient than the cut-and-choose schemes, but increases the computational burden of the parties. In particular, the protocol of [JS07] requires $O(|C|)$ public-key operations while the cut-and-choose schemes only require $O(m)$ public-key operations. As shown in experiments (e.g. see [MNPS04]) the public-key operations tend to be the computational bottle-neck in practice.

The idea of allowing the adversary to cheat as long as it will be detected with a reasonable probability was first considered in [FY92] under the term t-detectability. Work of [FY92] only considers honest majority and the definition is not simulation based. Canetti and Ostrovsky [CO99] consider *honest-looking adversaries* who may deviate arbitrarily form the protocol specification as long as the deviation cannot be detected. [AL07] introduce the notion of covert adversaries which is similar in nature to the previous works but strengthens them in several ways. The most notable are that it quantifies over all possible adversaries (as opposed to adversaries that behave in a certain way), and puts the burden of detection of cheating on the protocol, and not on the honest parties analyzing the transcript distribution later on.

2 Preliminaries

2.1 Definition of Security Against Covert Adversaries

Aumann and Lindell, [AL07], give a formal definition of security against covert adversaries in the *ideal/real simulation paradigm.* This notion of adversary lies somewhere between those of semi-honest and malicious adversaries. Loosely speaking, the definition provides the following guarantee: Let $0 \leq \epsilon \leq 1$ be a value (called the deterrence factor). Then any attempts to cheat by an adversary is detected by the honest parties with probability at least ϵ. Thus provided that ϵ is sufficiently large, an adversary that wishes not to get caught cheating will refrain from attempting to cheat, lest it be caught doing so. Furthermore, in the strongest version of security against covert adversaries introduced in [AL07], the adversary will not learn any information about the honest parties' inputs if he gets caught. What follows next is the strongest version of their definition (which is what we use as the security definition for all of our protocols) and is directly taken from [AL07]. The executions in the real and ideal model are as follows:

Execution in the real model. Let the set of parties be $P_1, \ldots, P_n$ and let $\mathcal{I} \subset [n]$ denote the indices of corrupted parties, controlled by an adversary $\mathcal{A}$. We consider the real model in which a real n-party protocol π is executed (and there exist no trusted third party). In this case, the adversary $\mathcal{A}$ sends all messages in place of corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of π.

Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ be an n-party functionality where $f = (f_1, \ldots, f_n)$, and let π be an n-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilist polynomial-time machine and let $\mathcal{I}$ be the set of corrupted parties. Then the real execution of π on inputs $\bar{x}$, auxiliary input z to $\mathcal{A}$ and security parameter s, denoted $REAL_{\pi,\mathcal{A}(z),\mathcal{I}}(\bar{x}, s)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Execution in the Ideal Model. Let $\epsilon : \mathcal{N} \rightarrow [0, 1]$ be a function. Then the ideal execution with ϵ proceeds as follows.
Inputs: Each party obtains an input; the i^{th} party's input is denoted by x_i; we assume that all inputs are of the same length m. The adversary receives an auxiliary-input z.
Send inputs to trusted party: Any honest party P_j sends its received input x_j to the trusted party. The corrupted parties, controlled by $\mathcal{A}$, may either send their received input or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on x_i for $i \in \mathcal{I}$ and the auxiliary input z. Denote the vector of inputs sent to the trusted party by $\bar{w}$.
Abort Options: If a corrupted party sends $w_i = \mathsf{abort}_i$ to the trusted party as its input, then the trusted party sends abort_i to all of the honest parties and

halts. If a corrupted party sends $w_i = \mathsf{corrupted}_i$ as its input to the trusted party, then the trusted party sends $\mathsf{corrupted}_i$ to all of the honest parties and halts.
Attempted cheat option: If a corrupted party sends $w_i = \mathsf{cheat}_i$ to the trusted party as its input, then:

1. With probability $1 - \epsilon$, the trusted party sends $\mathsf{corrupted}_i$ to the adversary and all of the honest parties.
2. With probability ϵ, the trusted party sends undetected and all of the honest parties inputs $\{x_j\}_{j \notin \mathcal{I}}$ to the adversary. The trusted party asks the adversary for outputs $\{y_j\}_{j \notin \mathcal{I}}$, and sends them to the honest parties.

The ideal execution then ends at this point. If no w_i equals abort_i, $\mathsf{corrupted}_i$ or cheat_i the ideal execution continues below.
Trusted party answers adversary: The trusted party computes $(f_1(\bar{w}), \ldots, f_m(\bar{w}))$ and sends $f_i(\bar{w})$ to $\mathcal{A}$, for all $i \in \mathcal{I}$.
Trusted party answers honest parties: After receiving its outputs, the adversary sends either abort_i for some $i \in \mathcal{I}$ or continue to the trusted party. If the trusted party receives the continue then it sends $f_i(\bar{w})$ to all honest parties $P_j (j \notin \mathcal{I})$. Otherwise, if it receives abort_i for some $i \in \mathcal{I}$, it sends abort_i to all honest parties.
Outputs: An honest party always outputs the messages it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{x_i\}_{i \in \mathcal{I}}$ and messages obtained from the trusted party.

The output of honest parties and the adversary in an execution of the above model is denoted by $IDEAL^{\epsilon}_{f,S(z),\mathcal{I}}(\bar{x}, s)$ where s is the statistical security parameter.

Definition 1 *Let f, π, ϵ be as described above. Protocol π is said to* securely compute f in the presence of covert adversaries with $\epsilon-$deterrence *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exist a non-uniform probabilistic polynomial-time adversary S for the ideal model such that for every $\mathcal{I} \subseteq [n]$, every balanced vector $\bar{x} \in (\{0,1\}^*)^n$, and every auxiliary input $z \in \{0,1\}^*$:*

$$IDEAL^{\epsilon}_{f,S(z),\mathcal{I}}(\bar{x}, s) \stackrel{c}{\equiv} REAL_{\pi,\mathcal{A}(z),\mathcal{I}}(\bar{x}, s)$$

3 The Two Party Case

3.1 Efficient Two Party Computation for Covert Adversaries

Aumann and Lindell [AL07] design an efficient two-party computation protocol secure against covert adversaries. In their protocol, two parties P_1 and P_2 wish to securely compute a circuit C that computes a function f on parties private inputs. The high level idea of their protocol is that party P_1 computes t garbled circuits[4], and sends them to party P_2. P_2 then randomly chooses one circuit to compute and asks P_1 to reveal the secrets of the remaining $(t-1)$ circuits. This ensures that a cheating P_1 gets caught with probability at least equal to $1-1/t$. There are other subtleties in order to deal with parties' inputs and to achieve simulation-based security. We will go into more detail regarding these subtleties later in this section. Aumann and Lindell also design a special and highly efficient oblivious transfer protocol secure against covert adversaries which makes their solution even more practical. The efficiency of their protocol can be summarized in the following statement ($|C|$ is the circuit size, m is the input size and s is the security parameter):

Theorem 1 *([AL07]) There exist a two-party computation protocol secure against covert adversaries with deterrence value $1-1/t$ such that the protocol runs in a constant number of rounds, and requires $O(t|C|+tsm)$ communication between the two players.*

Our Protocol We now design a secure two-party computation protocol in presence of covert adversaries for which the deterrence probability $1-1/t$, for any value of t polynomial in the security parameter, comes almost for free in terms of the *communication complexity of the protocol* (assuming the circuit being evaluated is large enough). In the remiander of the paper, we assume familiarity with the Yao's garbled circuit protocol.

We first observe that for the simulation-based proof of the protocol to go through and for the simulator to be able to extract corrupted P_2's inputs, it is not necessary to run the complete oblivious transfers early in the protocol for all the garbled circuits. Instead, it is enough to go as far in the steps of the OTs as is necessary for party P_2 to be committed to his input bits while party P_1 is still free to choose his inputs to the OT. Parties then postpone the remaining steps of the OTs until later in the protocol when one circuit among the t garbled circuits is chosen to be evaluated. With some care, this leads to asymptotic improvement in communication complexity of our protocol.

To achieve further improvement in communication complexity, we take a different approach to constructing the garbled circuit. In order to compute a garbled circuit (and the commitments for input keys), party P_1 generates a short random seed and feeds it to a pseudorandom generator in order to generate the necessary randomness. He then uses the randomness to construct the garbled circuit and the necessary commitments. When the protocol starts, party P_1 sends to P_2 only a hash of each garbled circuit using a collision-resistant hash function. Later in the protocol, in order to expose the secrets of each circuit, party P_1 can simply send the seeds corresponding to that circuit to P_2, and not the whole opened circuit. In the full version of this paper, we describe in more detail, how to generate the garbled circuit in this way.

[4] The garbled circuits are constructed according to Yao's garbled circuit protocol(see [LP04] for a detailed explanation).

Before describing the details of our protocol, it is helpful to review a trick introduced by [LP07] for preventing a subtle malicious behavior by a corrupted P_1. For instance, during an oblivious transfer protocol, a corrupted P_1 can use an invalid string for the key associated with value 0 for P_2's input bit but a valid string for the key associated with 1. An honest P_2 is bound to abort if any of the keys he receives are invalid. But the action P_2 takes reveals his input bit to P_1. To avoid this problem, we use a circuit that computes the function $g(x_1, x_2^1, \ldots, x_2^s) = f(x_1, \oplus_{i=1}^{s} x_2^i)$ instead of a circuit that directly computes f. For his actual input x_2, party P_2 chooses s random inputs $x_2^1, \ldots, x_2^s$ such that $x_2 = x_2^1 \oplus \ldots \oplus x_2^s$. This solves the problem since for P_1 to learn any information about P_2's input he has to send invalid keys for all s shares. But, if P_1 attempts to give invalid key for all s shares of P_2's input, he will get caught with exponentially high probability in s. We are now ready to describe our protocol. We borrow some of our notations from [LP04] and [AL07].

The Protocol

Party P_1's input: x_1
Party P_2's input: x_2
Common input: Both parties have security parameter m; for simplicity let $|x_1| = |x_2| = m$. Parties agree on the description of a circuit C for inputs of length m that computes function f. P_2 chooses a collision-resistant hash function h. Parties agree on a pseudorandom generator G, a garbling algorithm *Garble*, a perfectly binding commitment scheme Com_b, and a deterrence probability $1 - 1/t$.

1. Parties P_1 and P_2 define a new circuit C' that receives $s+1$ inputs $x_1, x_2^1, \ldots, x_2^s$ each of length m, and computes the function $f(x_1, \oplus_{i=1}^{s} x_2^i)$. Note that C' has $m(s+1)$ input wires. Denote the input wires associated with x_1 by $w_1, \ldots, w_m$ and the input wires associated with x_2^i by $w_{im+1}, \ldots, w_{im+m}$ for $i = 1, \ldots, s$.
2. Party P_2 chooses $(s-1)$ random strings $x_2^1, \ldots, x_2^{s-1} \in_R \{0,1\}^m$ and defines $x_2^s = (\oplus_{i=1}^{s-1} x_2^i) \oplus x_2$. The value $z_2 = (x_2^1, \ldots, x_2^s)$ serves as P_2's new input of length sm to C'.
3. Parties perform the first four steps of the OT protocol of [AL07] for P_2's sm input bits (see the full version for more detail). [5]
4. Party P_1 generates t random seeds $s_1, \ldots, s_t$ of appropriate length and computes $GC_i = Garble(G, s_i, C')$ for $1 \le i \le t$ (see the full version of this paper for *Garble*() algorithm). He then sends $h(GC_1)), \ldots, h(GC_t)$ to P_2.
5. P_1 generates t random seeds $s'_1, \ldots, s'_t$ of appropriate length and computes $G(s'_i)$ from which he extracts the randomness $r_j^{b,i}$ (later used to construct

[5] Any other constant-round oblivious transfer protocol secure against covert adversaries with the property that– there exists an step in the protocol where P_2 is committed to his input while P_1 is still free to choose his input– can be used here as well.

a commitment) for every $1 \le i \le t$, every $j \in \{1, \ldots, sm + m\}$, and every $b \in \{0, 1\}$, and the random order for the commitments to keys for his own input wires (see next step). He then computes the commitments $c_j^{b,i} = Com_b(k_j^{b,i}, r_j^{b,i})$ for every $i \in \{1, \ldots, t\}$, every $j \in \{1, \ldots, sm + m\}$, and every $b \in \{0, 1\}$.

6. For every $1 \le i \le t$, P_1 computes two sets A_i and B_i, consisting of pairs of commitments. The order of each pair in B_i is chosen at random (using the randomness generated by $G(s_i'))$, but the order of each pair in A_i is deterministic, i.e., commitment to the key corresponding to 0 comes before the one corresponding to 1.

$$A_i = \{(c_{m+1}^{0,i}, c_{m+1}^{1,i}), \ldots, (c_{m+sm}^{0,i}, c_{m+sm}^{1,i})\}$$
$$B_i = \{(c_1^{0,i}, c_1^{1,i}), \ldots, (c_m^{1,i}, c_m^{0,i})\}$$

   P_1 then sends $h(A_1), \ldots, h(A_t)$ and $h(B_1), \ldots, h(B_t)$ to P_2.
7. P_2 chooses a random index $e \in_R \{0, 1\}^{\log(t)}$ and sends it to P_1.[6]
8. Let $O = \{1 \ldots e-1, e+1 \ldots t\}$. P_1 sends to P_2, s_i and s_i' for every $i \in O$. P_2 Computes $h(GC_i) = h(Garble(G, s_i, C'))$ for every $i \in O$ and verifies that they are equal to what he received from P_1. He also computes $G(s_i')$ to get the decommitment values for commitments in A_i and B_i for every $i \in O$. P_2 then uses the keys and decommitments to recompute $h(A_i)$ and $h(B_i)$ on his own for every $i \in O$, and to verify that they are equal to what he received from P_1. If not, it outputs $\mathsf{corrupted}_1$ and halts.
9. P_1 sends to P_2 the actual garbled circuit GC_e, and the sets of commitment pairs A_e and B_e (note that P_2 only held $h(GC_e)$, $h(A_e)$, and $h(B_e)$). P_1 also sends decommitments to the input keys associated with his input for the circuit.
10. P_2 checks that the values received are valid decommitments to the commitments in B_e (he can open one commitment in every pair) and outputs $\mathsf{corrupted}_1$ if this is not the case.
11. Parties perform steps 5 and 6 of the OT protocols (see the full version of this paper for details regarding how this is done). P_1's input to the OTs are random strings corresponding to the eth circuit. As a result, P_2 learns one of the two strings $(k_{i+m}^{0,e}||r_{i+m}^{1,e}, k_{i+m}^{1,e}||r_{i+m}^{1,e})$ for the i^{th} OT $(1 \le i \le sm)$.
12. P_2 learns the decommitments and key values for his input bits from the OTs' outputs. He checks that the decommitments are valid for the commitments in A_e and that he received keys corresponding to his correct inputs. He outputs $\mathsf{corrupted}_1$ if this is not the case. He then proceeds with computing the garbled circuit $C'(x_1, z_2) = C(x_1, x_2)$, and outputs the result. If the keys are not correct and therefore he cannot compute the circuit, he outputs $\mathsf{corrupted}_1$.
13. If at anytime during the protocol one of the parties aborts unexpectedly, the other party will output abort and halt.

[6] For simplicity we assume that t is a power of 2.

The general structure of our proof of security is the same as the proof in [AL07]. Due to lack of space details of the simulation are given in the full version of this paper. The following claim summarizes our result.

Claim. Assuming that h is a collision-resistant hash function, Com_b is a perfectly binding commitment scheme, and G is a pseudorandom generator, then the above protocol is secure against covert adversaries with deterrence value $1 - 1/t$. The protocol runs in a constant number of rounds, and requires $O(|C| + sm + t)$ communication between the two players.

3.2 Extension to General Secure Two Party Computation

Our technique of only sending a hash (using a collision resistant hash function) of circuits and commitments directly generalizes to to the case of secure two party computation in the standard malicious adversary model.

Almost all the existing works for defending Yao's garbled circuit protocol against malicious adversaries in an efficient way [MF06,LP07,Woo07] use the *cut-and-choose* techniques. More specifically, party P_1 sends t garbled circuits to P_2; half of the circuits are chosen at random and their secrets are revealed by P_1; the remaining circuits are evaluated and the majority value is the final output of the protocol. Additional mechanisms are used to verify input consistency and to force the parties to use the same input values for majority of the circuits. Using our new garbling method and sending hash of circuits instead of the circuits themselves (as discussed previously) we automatically improve efficiency of these protocols. By carefully choosing the number of hashed garbled circuits and the fraction of circuits that are opened, we can make the efficiency gain quite substantial. Please see the full version of this paper for more detail on good choices of parameters. Next we outline some of these efficiency gains through some concrete examples.

Efficiency in Practice For simplicity we demonstrate our improvements via comparison with the equality-checker scheme of [MF06] since a detailed analysis for it is available in [Woo07]. But, it is important to note that our techniques lead to similar improvements to all of the most-efficient protocols in the literature such as the expander-checker scheme of [Woo07] and the scheme proposed in [LP07] which also provides simulation-based security. Details of the modifications to the original equality-checker scheme are given in the full version of this paper.

By setting the parameters of the protocol (as we show in the full version of this paper), we can make the modified equality-checker (equality-checker-2) superior to the original one (equality-checker-1) in practice. The optimal choice of parameters depends on several factors such as the circuit size, the input size, and the size of the output of hash function. We work out some of these numbers in the full version to highlight the efficiency gained by using our techniques. Consider the following examples where the circuit are taken from [MNPS04]. Using those numbers, for a circuit that compares two 32-bit integers using 256 gates, our protocols roughly lead to factor of 12 improvement in communication complexity for the same probability of undetected cheating, and for a circuit that computes the median of two sorted arrays of ten 16-bit integers, with 4383 gates, we gain at least a factor of 30 improvement.

4 The Multi Party Case

We construct a multi party computation protocol secure against covert adversaries for a given deterrence parameter $1 - \frac{1}{t}$. Let there be n parties denoted by $P_1, \ldots, P_n$. The basic idea of the protocol is as follows. The parties run t parallel sessions, each session leading to the distributed generation of one garbled circuit. These sessions in the protocol are called the "garbled circuit generation sessions" (or GCG sessions in short). The protocol employed to generate these garbled circuits in the GCG sessions is a protocol secure only against semi honest adversaries and is based on the constant round BMR construction [BMR90]. Instead of employing zero knowledge proofs to go from semi-honest security to malicious security, we employ cut and choose techniques where the parties ensure the honesty of each other in $t-1$ random GCG sessions. This is done by generating a shared challenge string which is used to select the one GCG session whose garbled circuit will be used for actual computation. The parties are required to reveal the (already committed) randomness used for every other GCG session. For a party, given the randomness and the incoming messages, the outgoing messages become deterministic. Hence the whole transcript of a GCG session can be checked (given randomness used by all the parties in this session) and any deviations can be detected.

The main problem which we face to turn this basic idea into a construction is that the secret inputs of the honest parties might be leaked since an adversarial party might deviate arbitrarily from the protocol in any GCG session (and this deviation is not detected until all the sessions have finished). This is because the distributed garbled circuit generation ideas in the BMR construction [BMR90] make use of the actual inputs of the honest parties (so that for each input wire, parties have the appropriate key required to evaluate the resulting garbled circuit). To solve this problem, we modify the BMR construction "from the inside" to enable these GCG sessions execute without using the inputs of the parties. Our modifications also allow the parties to check honesty of each other in these sessions without revealing their individual inputs (while still allowing the simulator to be able to extract these inputs during the proof of security).

4.1 Building Blocks

One of the building blocks of our protocol is a secure function evaluation protocol which is secure against honest-but-curious adversaries, and whose round complexity is proportional to the multiplicative depth of the circuit being evaluated (over $\mathbb{Z}_2 = GF(2)$). A textbook protocol such as that given by Goldreich [Gol04] (which is a variant of the semi-honest GMW protocol [GMW87]) suffices. We remark that this protocol will be used only to evaluate very short and simple circuits (such as computing XOR of a few strings).

We also need several subprotocols which are secure against standard (not only covert) malicious adversaries. We summarize these here:

– **Simulatable Coin Flipping From Scratch (CoinFlipPublic):**
  This protocol emulates the usual coin-flipping functionality [Lin01] in the presence of arbitrary malicious adversaries. In particular, a simulator who controls a single player can control the outcome of the coin flip.

The remaining primitives assume the availability of a common random string σ. We assume that these primitives implement the corresponding ideal functionality in the CRS model.

– **Simultaneous commitment ($\mathsf{Commit}_\sigma(x_1, ..., x_n)$):** Every player chooses a value x_i and commits to it. At the end of the protocol, the vector of commitments is known to all parties. The commitments are such that a simulator having trapdoor information about the CRS σ can extract the committed values.
– **Open commitments ($\mathsf{OpenCom}_\sigma$):** Players simultaneously open their commitments over the broadcast channel.
  For the simulation to work, this protocol needs to be simulation-sound, in the following sense: if the simulator is controlling a subset of cheating players P_i, $i \in I_{sim}$, then he should be able to output a valid simulation in which all honest players lie about their committed values yet all cheating players are constrained to tell the truth or be caught.
– **Committed Coin Flipping ($\mathsf{CommitedCoinFlipPublic}_\sigma$ and $\mathsf{CommittedCoinFlip}_\sigma\mathsf{To}P_i$):**
  Generates a commitment to a random string such that all players are committed to shares of the coin. In the second variant, P_i learns the random string and is committed to it.
– **Open coin:**
  Opens a committed coin to all players over the broadcast channel. The simulator should be able to control the coin flip.

These primitives can be implemented very efficiently under several number-theoretic assumptions. For concreteness, we have described efficient instantiations based on the DDH assumption in the full version of this paper. These are summarized here.

Lemma 1. *Suppose the Decisional Diffie-Hellman problem is hard in group G. There exist secure implementations of the protocols above. The CRS protocols (Commit_σ, $\mathsf{OpenCom}_\sigma$, $\mathsf{CommitedCoinFlipPublic}_\sigma$, $\mathsf{CommittedCoinFlip}_\sigma\mathsf{To}P_i$) require $O(n\ell+n^2k)$ bits of communication each, and a shared CRS of length $2n+1$ group elements. Here k is the bit length of the elements of the group G, and ℓ is the bit length of the strings being generated, committed, or opened. Generating a CRS of length ℓ bits via* CoinFlipPublic *requires $O(n^2 \log(n)k + n\ell)$ bits of communication and $O(\log n)$ rounds.*

4.2 Main Multiparty Protocol

We now turn to the protocol itself. Let C be a circuit corresponding to the function $f(x_1, x_2, \ldots, x_n)$ which the parties wish to jointly compute. We denote the total number of wires (including the input and output wires) in C by W, each having index in the range 1 to W . Let F and G be pseudorandom generators with seed length s (here s is the security parameter). The parties run the following protocol.

Stage 0 Collectively flip a single string σ having length $poly(s)$. The string σ is used as a CRS for the commitment and coin-flipping in the remaining stages of the protocol.

$$\sigma \leftarrow \mathsf{CoinFlipPublic}$$

Stage 1 The parties generate the commitment to a shared challenge random string $e \in [t]$

$$e \leftarrow \mathsf{CommitedCoinFlipPublic}_\sigma$$

The challenge e will later be used to select which of the GCG sessions (out of the t sessions) will be used for actual computation. The parties will be required to show that they were honest in all other GCG sessions (by revealing their randomness).

Stage 2 For each $i \in [n]$ and $S \in [t]$, collectively flip coins $r_i[S]$ of length s and open the commitment (and decommitment strings) to P_i only:

$$r_i[S] \leftarrow \mathsf{CommittedCoinFlip}_\sigma\mathsf{To}P_i$$

Thus, a party P_i obtains a random string $r_i[S]$ for every session $S \in [t]$. All other parties have obtained commitment to $r_i[S]$. The random string $r_i[S]$ can be expanded using the pseudorandom generator F. It will be used by P_i for the following:

- To generate the share $\lambda_i^w[S] \in \{0,1\}$ of the *wire mask* $\lambda^w[S]$ (in Stage 3 of our protocol) for every wire w in the garbled circuit $GC[S]$ to be generated in session S. Recall that in a garbled circuit $GC[S]$, for every wire w, we have two *wire keys* (denoted by $k^{w,0}[S]$ and $k^{w,1}[S]$): one corresponding to the bit on wire w being 0 and the other to bit being 1 (during the actual evaluation of the garbled circuit, a party would only be able to find one of these keys for every wire). The wire mask determine the correspondence between the two wire keys and the bit value, i.e., the key $k^{w,b}[S]$ corresponds to the bit $b \oplus \lambda^w[S]$.
- To run the GCG session S (i.e., Stage 4 of our protocol). Note that we generate the wire masks for the garbled circuits in stage 3 (instead of 4) to enable the parties to run stage 4 without using their inputs.

Stage 3 Every player P_i is responsible for a subset of the input wires J_i, and holds an input bit x^w for each $w \in J_i$. For every $w \in J_i$, and session S, P_i computes $I^w[S] = x^w \oplus \lambda_i^w[S]$. For each S, players simultaneously commit to the value I^w for each of their input wires (each input wire is committed to by exactly one player):

$\{COM(I^w[S])\ :\ \text{input wires}\ \ w\} \leftarrow \mathsf{Commit}_\sigma\Big(\{I^w[S]\ :\ S \in \{1,...,t\},\ w \in$ input wires $\}\Big)$
Recall that exactly one of the sessions will be used for actual secure function evaluation. In that session, the above commitment will be opened and $x^w \oplus \lambda_i^w[S]$ will be revealed (however $\lambda_i^w[S]$ will remain hidden). In rest of sessions where the garbled circuit generated will be opened and checked completely by all the parties, the wire mask share $\lambda_i^w[S]$ will be revealed (since its a part of the garbled circuit description and generated using randomness $r_i[S]$). However the above commitment to $x^w \oplus \lambda_i^w[S]$ will *not* be opened for those sessions. This ensures the secrecy of the input x^w (while still allowing to simulator to extract it in our proof of security).

Stage 4 This is the stage in which the parties run t parallel garbled circuit generation session. This stage is based on the BMR construction but does not make use of the inputs of the parties. Each session in this stage can be seen as an independent efficient protocol (secure against honest but curious adversaries) where:

– In the beginning, the parties already hold shares of the wire masks $\lambda_i^w[S]$ to be used for the garbled circuit generation (as opposed to generating these wire masks in this protocol itself).
– In the end, the parties hold a garbled circuit $GC[S]$ for evaluating the function f. Furthermore, each party also holds parts of the wire keys for input wires (such that when for all input wires, all the parts of the appropriate wire key are broadcast, the parties can evaluate the garbled circuit; which key is broadcast is decided by the openings of the commitments of stage 3).

We now describe this stage in more detail.

1. P_i broadcasts the wire mask shares $\lambda_i^w[S]$ for all input wires belonging to other players (i.e., for w not in J_i), and for all output wires. Thus only the masks for P_i's inputs, and for internal wires, remain secret from the outside world . Note that $\lambda^w[S] = \bigoplus_{i=1}^n \lambda_i^w[S]$ is the wire mask for wire w. Each player holds shares of the wire masks.
2. For every wire w of the circuit C, P_i generates two random *key parts* $k_i^{w,0}[S]$ and $k_i^{w,1}[S]$. The full wire keys are defined as the concatenation of the individual key parts. That is, $k^{w,0}[S] = k_1^{w,0}[S] \circ \ldots \circ k_n^{w,0}[S]$ and $k^{w,1}[S] = k_1^{w,1}[S] \circ \ldots \circ k_n^{w,1}[S]$.
3. Recall that for every gate in the circuit, the wire keys of incoming wires will be used to encrypt the wire keys for outgoing wires (to construct what is called a *gate table*). However it is not desirable to use a regular symmetric key encryption algorithm for this purpose. The reason is that the gate tables will be generated by using a (honest but curious) secure function evaluation protocol (see next step) and the complexity of the circuit to be evaluated will depend upon the complexity of the encryption algorithm. To avoid this problem, the parties locally expand their key parts into large strings (and then later simply use a one time

pad to encrypt). More precisely, P_i expands the key parts $k_i^{w,0}[S]$ and $k_i^{w,1}[S]$ using the pseudorandom generator G to obtain two new keys, i.e.,$(p_i^{w,\ell}[S], q_i^{w,\ell}[S]) = G(k_i^{w,\ell}[S])$, for $\ell \in \{0,1\}$. Each of the new keys has length $n|k_i^{w,\ell}[S]|$ (enough to encrypt a full wire key).

4. The players then run a Secure Function Evaluation protocol secure against *honest-but-curious* adversaries to evaluate a simple circuit to generate the *gate tables*. This stage is inspired by a similar stage of the Beaver et al. protocol [BMR90]. This is the step that dominates the computation and communication complexity of our construction. However as opposed to BMR, *the underlying multi-party computation protocol used here only needs to be secure against semi-honest adversaries.* More details follow. For every gate g in the circuit C, define a gate table as follows. Let a, b be the two input wires and c be the output wire for the gate g, and denote the operation performed by the gate g by $\otimes$ (e.g. AND, OR, NAND, etc). Before the protocol starts, P_i holds the following inputs: $p_i^{a,\ell}[S], q_i^{a,\ell}[S], p_i^{b,\ell}[S], q_i^{b,\ell}[S], k_i^{c,\ell}[S]$ where $\ell \in \{0,1\}$ along with shares $\lambda_i^a[S], \lambda_i^b[S], \lambda_i^c[S]$ of wire masks $\lambda^a[S], \lambda^b[S], \lambda^c[S]$. P_i runs the protocol along with other parties to compute the following gate table:

$$\begin{aligned}
A_g &= p_1^{a,0}[S] \oplus \ldots \oplus p_n^{a,0}[S] \oplus p_1^{b,0}[S] \oplus \ldots \oplus p_n^{b,0}[S] \\
&\quad \oplus \begin{cases} k_1^{c,0}[S] \circ \ldots \circ k_n^{c,0}[S] & \text{if } \lambda^a[S] \otimes \lambda^b[S] = \lambda^c[S] \\ k_1^{c,1}[S] \circ \ldots \circ k_n^{c,1}[S] & \text{otherwise} \end{cases} \\
B_g &= q_1^{a,0}[S] \oplus \ldots \oplus q_n^{a,0}[S] \oplus p_1^{b,1}[S] \oplus \ldots \oplus p_n^{b,1}[S] \\
&\quad \oplus \begin{cases} k_1^{c,0}[S] \circ \ldots \circ k_n^{c,0}[S] & \text{if } \lambda^a[S] \otimes \overline{\lambda^b[S]} = \lambda^c[S] \\ k_1^{c,1}[S] \circ \ldots \circ k_n^{c,1}[S] & \text{otherwise} \end{cases} \\
C_g &= p_1^{a,1}[S] \oplus \ldots \oplus p_n^{a,1}[S] \oplus q_1^{b,0}[S] \oplus \ldots \oplus q_n^{b,0}[S] \\
&\quad \oplus \begin{cases} k_1^{c,0}[S] \circ \ldots \circ k_n^{c,0}[S] & \text{if } \overline{\lambda^a[S]} \otimes \lambda^b[S] = \lambda^c[S] \\ k_1^{c,1}[S] \circ \ldots \circ k_n^{c,1}[S] & \text{otherwise} \end{cases} \\
D_g &= q_1^{a,1}[S] \oplus \ldots \oplus q_n^{a,1}[S] \oplus q_1^{b,1}[S] \oplus \ldots \oplus q_n^{b,1}[S] \\
&\quad \oplus \begin{cases} k_1^{c,0}[S] \circ \ldots \circ k_n^{c,0}[S] & \text{if } \overline{\lambda^a[S]} \otimes \overline{\lambda^b[S]} = \lambda^c[S] \\ k_1^{c,1}[S] \circ \ldots \circ k_n^{c,1}[S] & \text{otherwise} \end{cases}
\end{aligned}$$

This circuit has multiplicative depth 2. If we use the honest-but-curious SFE protocol from [Gol04], this stage requires a constant number of rounds. At the end of this phase, for each session S, the parties hold a garbled circuit $GC[S]$ (which consists of the gate tables as generated above, along with the wire masks $\lambda^w[S]$ for each *output wire* w).

Stage 5 The parties now open the challenge e generated in Step 3, using $\mathsf{OpenCom}_\sigma$.

Stage 6 For each session $S \neq e$, each party P_i opens the commitment to $r_i[S]$ generated in Step 1. Given $r_1[S], \ldots, r_n[S]$, all the wire mask shares and the protocol of Stage 4.2 become completely deterministic. More precisely, each player can regenerate the transcript of Stage 4.2, and can thus verify that all parties played honestly for all sessions $S \neq e$. If P_i detects a deviation from the honest behavior, it aborts identifying the malicious party P_j who deviated.

Note that the only point so far where the parties were required to use their inputs is Stage 3 (where P_i committed to $x^w \oplus \lambda_i^w[S]$ for all $w \in J_i$). However these commitments were not used in any other stage. Hence, since these commitments have not yet been opened nor used anywhere else, if the players abort at this stage then no information is learned by the adversary.

Once the parties successfully get past this stage without aborting, we have a guarantee that the garbled circuit $GC[e]$ was correctly generated except with probability $\frac{1}{t}$. Thus, $\frac{1}{t}$ bounds the probability with which an adversary can cheat successfully in our protocol.

Stage 7 For all input wires $w \in J_i$, P_i now opens the commitments $COM^w[e]$ (see Stage 3) using $\mathsf{OpenCom}_\sigma$, thus revealing $I^w = \lambda_i^w[e] \oplus x^w$. Set $L^w = I^w \oplus \bigoplus_{j=1}^{i-1} \lambda_j^w[e] \oplus \bigoplus_{j=i+1}^{n} \lambda_j^w[e]$ (where $\lambda_j^w[e]$ was broadcast in stage 4(a)), i.e., $L^w = \lambda^w[e] \oplus x^w$. Every party P_ℓ, $1 \leq \ell \leq n$ broadcasts the key parts $k_\ell^{w,L^w}[e]$.

Stage 8 P_i now has the garbled circuit $GC[e]$ as well the wire keys $k^{w,L^w}[e] = k_1^{w,L^w}[e] \circ \ldots \circ k_n^{w,L^w}[e]$ for all input wires w of the circuit. Hence P_i can now evaluate the garbled circuit on its own in a standard manner to compute the desired function output $C(x_1, x_2, \ldots, x_n)$. For more details on how the garbled circuit $GC[e]$ is evaluated, see [BMR90].

The following theorem summarizes our result. See the full version of this paper for the analysis of our construction.

Theorem 2 *If the coin-flipping and commitment primitives are secure against malicious adversaries and the SFE scheme is secure against honest-but-curious adversaries, then the above construction is secure in the presence of covert adversaries with $1 - \frac{1}{t}$ deterrence.*

If we instantiate the coin-flippping and commitment primitives as in Lemma 1, and use the SFE scheme of [Gol04], then the protocol above requires $O(\log n)$ rounds and a total of $O(n^3 ts|C|)$ bits of communication to evaluate a boolean circuit of size $|C|$, where s is the security parameter (the input size of a pseudorandom generator). The computational complexity is the same up to polylogarithmic factors.

If we use the constant-round coin-flipping protocols of Katz et al. [KOS03] or Pass [Pas04], then the protocol above runs in constant rounds, but requires substantially slower (though still polynomial) computations.

The protocol above is the first multiparty protocol we know of which is tailored to covert adversaries. As a point of comparison, to our knowledge the most efficient protocol secure against *malicious* adversaries that tolerates up to $n-1$ cheaters is that of Katz et al. [KOS03]. The running time of the KOS protocol is dominated by the complexity of proving statements *about* circuits of size $O(n^3 s|C|)$ (this is the cost incurred by compiling an honest-but-curious SFE protocol). In contrast, our protocol runns in time $\tilde{O}(n^3 st)$. Thus, the contribution of this protocol can be seen as relating the complexity of security against covert adversaries to security against honest-but-curious adversaries:

$$\begin{aligned} &\text{Cost of deterrence } 1 - \frac{1}{t} \text{ against covert adversaries} \\ &\qquad\qquad \lesssim \; t \cdot \Big(\text{Cost of honest-but-curious garbled circuit generation}\Big) \end{aligned}$$

References

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference, TCC*, 2007.

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *FOCS*, pages 345–355. IEEE Computer Society, 2002.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of ACM STOC, pages 1-10*, 1988.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multi-party unconditionally secure protocols. In *Proceedings of ACM STOC, pages 11-19*, 1988.

[CO99] Ran Canetti and Rafail Ostrovsky. Secure computation with honest-looking parties: What if nobody is truly honest? (extended abstract). In *STOC*, pages 255–264, 1999.

[CR87] Benny Chor and Michael O. Rabin. Achieving independence in logarithmic number of rounds. In *PODC*, pages 260–268, 1987.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.

[FY92] Matthew Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). pages 699–710, 1992.

[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] Oded Goldreich. *Foundation of Cryptography, Volume II: Basic Applications*. Cambridge University Press, 2004.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In Jon M. Kleinberg, editor, *STOC*, pages 99–108. ACM, 2006.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, 2007.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 578–595. Springer, 2003.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 171–189, London, UK, 2001. Springer-Verlag.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography Conference, PKC*, 2006.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system, 2004.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *STOC*, pages 232–241. ACM, 2004.

[Pin03] Benny Pinkas. Fair secure two-party computation. In *Eurocrypt '2003 Proceedings*, pages 87–105. Springer-Verlag, 2003.

[Woo07] David P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *EUROCRYPT*, 2007.

[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.