Covert Two-Party Computation

Luis von Ahn
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
biglou@cs.cmu.edu

Nicholas Hopper
University of Minnesota
200 Union St SE Rm 4-192
Minneapolis, MN 55455
hopper@cs.umn.edu

John Langford
TTI Chicago
1427 East 60th St, 2nd Floor
Chicago, IL 60637
jl@tti-c.org

ABSTRACT

We introduce *covert two-party computation*, a stronger notion of security than standard secure two-party computation. Like standard secure two-party computation, covert two-party computation allows Alice and Bob, with secret inputs x_A and x_B respectively, to compute a function $f(x_A, x_B)$ without leaking any additional information about their inputs. In addition, covert two-party computation guarantees that even the existence of a computation is hidden from all protocol participants unless the value of the function mandates otherwise. This allows the construction of protocols that return $f(x_A, x_B)$ only when it equals a certain value of interest (such as "Yes, we are romantically interested in each other") but for which *neither party can determine whether the other even ran the protocol whenever $f(x_A, x_B)$ is not a value of interest.* Since existing techniques for secure function evaluation always reveal that both parties participate in the computation, covert computation requires the introduction of new techniques based on provably secure steganography. We introduce security definitions for covert two-party computation and show that this surprising notion can be achieved by a protocol given the Decisional Diffie-Hellman assumption in the "honest but curious" model. Using this protocol as a subroutine, we present another protocol which is fair and secure against malicious adversaries in the Random Oracle Model — unlike most other protocols against malicious adversaries, this protocol does not rely on zero-knowledge proofs (or similar cut-and-choose techniques), because they inherently reveal that a computation took place. We remark that all our protocols are of comparable efficiency to protocols for standard secure two-party computation.

Categories and Subject Descriptors

F.m [**Theory of Computation**]: Miscellaneous; E.3 [**Data**]: Data Encryption

General Terms

Algorithms, Security, Theory

Keywords

Steganography, Two-Party Computation, Covert Computation, Fair two-party computation

1. INTRODUCTION

Secure two-party computation (or SFE, for secure function evaluation) allows Alice and Bob to evaluate a function of their secret inputs so that neither learns anything other than the output of the function. A real-world example often used to illustrate the applications of this primitive is when Alice and Bob wish to determine if they are romantically interested in each other. Secure two-party computation allows them to do so without revealing their true feelings *unless they are both attracted.* By securely evaluating the AND of the bits representing whether each is attracted to the other, both parties can learn if there is a match without risking embarrassment: if Bob is not interested in Alice, for instance, the protocol does not reveal whether Alice is interested in him.

However, although often used to illustrate the concept, this example is not entirely logical. The very use of two-party computation already reveals possible interest from one party: "would you like to determine if we are both attracted to each other?"

A similar limitation occurs in a variety of other applications where the very use of the primitive raises enough suspicion to defeat its purpose. To overcome this limitation we introduce *covert two-party computation*, which, unless the output of the function is favorable to both parties, *hides the fact that a computation even took place.* The computation is hidden in that each party cannot determine if the other went along with the computation or simply was communicating as they normally do. More specifically, for a given set Y of interesting values, covert two-party computation guarantees the following (in addition to leaking no additional knowledge about the individual inputs): (A) no outside eavesdropper can determine whether the two parties are performing the computation or simply communicating as they normally do; (B) When $f(x_A, x_B) \in Y$, the result of the computation is revealed; but (C) neither party can determine whether the other even ran the protocol whenever $f(x_A, x_B) \notin Y$. Thus whether or not a computation took place is revealed based on the output of the function.

1.1 Some Applications

Among the many applications of covert two-party computation we mention the following as examples:

- **Covert Authentication.** Imagine that Alex works for CIA and Bob works for Mossad (the Israeli intelligence agency). Both have infiltrated a single terrorist cell but neither knows that the other is also an undercover agent. If they can discover their "mutual interest" they could pool their efforts; thus both should be looking for potential collaborators. On the other hand, suggesting something out of the ordinary is happening to a normal member of the cell would likely be fatal. Running a covert computation in which both parties' inputs are their signed credentials and the result is revealed only if they are allies will allow Alex and Bob to authenticate each other. If Bob is NOT an ally, he will not know that Alex was even asking for authentication, and vice-versa. Furthermore, the authentication would happen without anybody else being able to notice that something out of the ordinary happened.

It is important that *the existence of the computation is revealed only after the parties are mutually authenticated*, since asking Charlie, a terrorist, to execute *any* SFE protocol could result in Alex being killed. (Similar situations occur in, e.g., planning a *coup d'etat* or constructing a zombie network.)

- **Dating.** As hinted above, covert two-party computation can be used to properly determine if two people are romantically interested in each other. It allows a person to approach another and perform a computation hidden in their normal-looking messages such that: (1) if both are romantically interested in each other, they both find out; (2) if none or only one of them is interested in the other, neither will be able to determine that a computation even took place.

  Of course, ordinary social interactions do not require such discretion[1]. But if the computation further involves comparing some "unusual" dating preferences or if one of the parties is already involved with somebody else, it could be embarrassing or detrimental to reveal the need to run the protocol at all.

  To prevent one party from lying to determine if the other is interested, Alice and Bob could use a function which outputs a non-repudiable "certificate" of mutual interest; but even in this case it is important to guarantee that *both* obtain the result. If one of the parties can get the result while ensuring that the other one doesn't, this party would be able to learn the other's input by pretending he is romantically interested; there would be no harm for him in doing so since the other would never see the result. Two-party computation protocols in which either both parties get the result at roughly the same time or neither of them does are called "fair." We will present a protocol for fair covert two-party computation.

- **Cooperation between competitors.** Imagine that Alice and Bob are competing online retailers and both are being compromised by a sophisticated hacker. Imagine also that because of the volume of their logs, neither Alice nor Bob can draw a reliable inference about the location of the hacker, and statistical analysis indicates that about twice as many attack events are required to isolate the hacker. Thus if Alice and Bob were to compare their logs, they could solve their problem. But if Alice admits she is being hacked and Bob is not, he will certainly use this information to overtake her customers, and vice-versa. Using covert computation to perform the log analysis online can break this impasse.

  If Alice is concerned that Bob might fabricate data to try and learn something from her logs, the computation could be modified so that when an attacker is identified, the output is both an attacker and a signed contract stating that Alice is due a prohibitively large fine (for instance, \$1 Billion US) if she can determine that Bob falsified his log, and vice-versa.

  Similar situations occur whenever cooperation might benefit mutually distrustful competitors. For example: negotiations for a merger, contract negotiations including terms which are illegal, etc.

- **Cheating in card games.** Suppose two parties playing a card game want to determine whether they should cheat. Each of them is self-interested, so cheating should not occur unless both players can benefit from it. Using covert two-party computation with both players' hands as input allows them to compute if they have an opportunity to benefit from cheating while guaranteeing that: (1) neither player finds out whether the other attempted to cheat unless they can both benefit from it; (2) none of the other players can determine if the two are secretly planning to collude.

  Hiding the desire to cheat (i.e., hiding that the function is being evaluated) is especially useful because such desire is greatest when a player has a weak hand, so even revealing this desire already reveals something about the player's input.

This list of "colorful" applications is not intended to be exhaustive; we assume that the reader can see the potential benefit of this new primitive.

1.2 Synchronization

A natural question to ask about covert computation is, "Don't the parties have to reveal that they want to use a covert protocol anyway?" This, however, is not the case: a protocol specification can include a public and well-known method for how and when to initiate the protocol. An example of such "synchronization" information could be: "if we will determine whether we both like each other, the computation will start with the first message exchanged after 5pm." Since such details are published as part of the protocol specification, there is no need for either party to indicate that they wish to compute anything at all: if Alice is interested in computing with Bob, she starts the protocol with her first message after 5pm.

Who knows what?

Given the guarantees that covert computation offers, it is important to clarify what the parties know and what they don't. We assume that both parties know a common circuit for the function that they wish to evaluate, that they know which role they will play in the evaluation, and that they know when to start evaluating the circuit if the computation is going to occur. Finally, we assume adversarial parties know all such details of the protocols we construct.

1.3 Contributions and Organization

The primary contribution of this paper is to introduce a new cryptographic primitive, *covert computation*, and to show that this primitive is feasible by giving a construction which can covertly realize any functionality. The existence of such a primitive is in itself surprising — you could be computing something without knowing if the other party is going along with the computation — and cannot be accomplished using standard techniques. We outline the technical obstacles in this section.

Hiding the Computation vs. Hiding the Function.

Notice that covert computation is not about hiding *what function* Alice and Bob are interested in computing, which

[1] except for the very shy

could be accomplished via standard SFE techniques. *Covert computation hides the fact that Alice and Bob are interested in computing a function at all.* This point is vital in the case of, e.g., covert authentication, where expressing a desire to do *anything* out of the ordinary could result in the death of one of the parties. In fact, we assume that the specific function to be computed (if any) is known to all parties. This is analogous to the difference in security goals between steganography (where the adversary is assumed to know which message, if any, is hidden and simply has to determine whether such a hidden message exists) and encryption, where the adversary is trying to decide what the hidden message is.

Can covert two-party computation be achieved by trivial composition of SFE protocols with steganography?

No. Steganographically encoding all messages of a standard secure computation protocol would yield a protocol for which no outside observer can determine whether it is being run, but would **not** guarantee that the participating parties themselves cannot tell that the protocol is being run. Covert two-party computation guarantees that the computation remains hidden from the participating parties (except for certain output values of the function being computed). Provably secure steganography [17], however, is an important step towards making covert two-party computation conceivable.

Security against malicious adversaries without Zero Knowledge.

Our basic protocol uses Yao's "garbled circuit" construction for SFE as a subroutine. Unfortunately Yao's protocol is either inherently unfair (only one party can obtain the result) or insecure against malicious adversaries (because the functionality of the circuit is concealed). Previous techniques for simultaneously obtaining security against malicious adversaries and fairness rely on zero-knowledge proofs (or similar cut-and-choose techniques) to prove that the "garbled" circuit computes the agreed-upon function. Such techniques cannot be applied in the case of covert computation, because they inherently reveal the computation.

Our solution uses a form of "reactive computation" in which the basic, unfair protocol is invoked many times as a subroutine; each time it gives an output which appears random but carries state forward to the next invocation. Since our final protocol is fair, this technique may be of independent interest; in some situations it is more efficient than any previous fair protocol for general SFE. In particular, suppose that k is a security parameter, n is the size of the inputs, $c_f(n)$ is the size of a circuit to evaluate the function f, $c_h(k, n)$ is the size of a circuit to evaluate a cryptographic hash function on $\max\{k, n\}$ bits, and $c_{ot}(k, l)$ is the communication complexity of OT_1^2 with l-bit strings and security parameter k. Then Yao's (unfair) protocol has communication complexity $O(kc_f(n) + nc_{ot}(k, k))$; our solution can be implemented with communication complexity $O(kc_f(n) + k^2 c_h(k, n) + (k^2 + n)\ c_{ot}(k, k))$; and the most efficient protocol for fair secure two-party computation known prior to our work has complexity $O(k^2 c_f(n) + nc_{ot}(k, k^2))$ [21] (we remark, however, that this protocol is secure in the standard model whereas our security proof is in the random oracle model).

Roadmap.

The high-level view of our presentation is as follows. First, we define "ordinary" or "innocent-looking" communications. Our protocols will generate messages that are indistinguishable from "ordinary" communications — so nobody can tell if the parties are performing a computation or just communicating innocently. The first protocol we present is a modification of Yao's "garbled circuit" two-party protocol. We show how to instantiate the encryption and oblivious transfer primitives in Yao's protocol to yield a complete protocol for two-party secure function evaluation that generates messages indistinguishable from uniform random bits. We then use provably secure steganography to transform this into a protocol that generates messages indistinguishable from "ordinary" communications. The protocol thus constructed, however, is not secure against malicious adversaries nor is it fair (since neither is Yao's protocol by itself). We therefore construct another protocol, which uses our modification of Yao's protocol as a subroutine, that satisfies fairness and is secure against malicious adversaries, in the Random Oracle Model.

Related Work.

Secure two-party computation was introduced by Yao [22]. Since then, there have been several papers on the topic and we refer the reader to a survey by Goldreich [12] for further references. Constructions that yield fairness for two-party computation were introduced by Yao [23], Galil et al. [11], Brickell et al. [6], and many others (see [21] for a more complete list of such references). The notion of covert computation, however, is completely new.

Notation.

We say a function $\mu : \mathbb{N} \to [0, 1]$ is *negligible* if for every $c > 0$, for all sufficiently large k, $\mu(k) < 1/k^c$. We denote the length (in bits) of a string or integer s by $|s|$ and the concatenation of string s_1 and string s_2 by $s_1||s_2$. We let U_k denote the uniform distribution on k bit strings. If $\mathcal{D}$ is a distribution with finite support X, we the *minimum entropy* of $\mathcal{D}$ is given by $H_\infty(\mathcal{D}) = \min_{x\in X}\{-\log_2 \Pr_{\mathcal{D}}[x]\}$. The *statistical distance* between two distributions $\mathcal{C}$ and $\mathcal{D}$ with joint support X is defined by $\Delta(\mathcal{C}, \mathcal{D}) = (1/2)\sum_{x\in X} |\Pr_{\mathcal{D}}[x] - \Pr_{\mathcal{C}}[x]|$. Two sequences of distributions, $\{\mathcal{C}_k\}_k$ and $\{\mathcal{D}_k\}_k$, are called *computationally indistinguishable*, written $\mathcal{C} \approx \mathcal{D}$, if for any probabilistic polynomial-time $\mathbf{A}$, the quantity $\mathbf{Adv}^{\mathbf{A}}_{\mathcal{C},\mathcal{D}}(k) = |\Pr[\mathbf{A}(\mathcal{C}_k) = 1] - \Pr[\mathbf{A}(\mathcal{D}_k) = 1]|$ is negligible in k.

2. BIDIRECTIONAL CHANNELS

We hide the communication patterns of two-party computation protocols in "ordinary" or "innocent-looking" messages. We define ordinary communication patterns and messages in a manner similar to the *channels* used by [17, 2, 9]. The main difference is that our channel is shared among two participants and messages sent by each participant might depend on previous messages sent by either one of them. To emphasize this difference, we use the term *bidirectional channel*.

Messages are drawn from a set D of *documents*. For simplicity we assume that time proceeds in discrete *timesteps*. Each party $P \in \{P_0, P_1\}$ maintains a history h_P, which represents a timestep-ordered list of all documents sent and received by P. We call the set of well-formed histories $\mathcal{H}$. We associate to each party P a family of probability distributions $\mathcal{B}^P = \{\mathcal{B}_h^P\}_{h\in\mathcal{H}}$ on D.

The communication over a bidirectional channel $\mathcal{B} = (D, \mathcal{H}, \mathcal{B}^{P_0}, \mathcal{B}^{P_1})$ proceeds as follows. At each timestep, each party P receives messages sent to them in the previous timestep, updates h_P accordingly, and draws a document $d \leftarrow \mathcal{B}_{h_P}^P$ (the draw could result in the empty message $\perp$, signifying that no action should be taken that timestep). The document d is then sent to the other party and h_P is updated. We assume for simplicity that all messages sent at a given timestep are received at the next one. Denote by $\mathcal{B}_{h_P}^P \neq\perp$ the distribution $\mathcal{B}_{h_P}^P$ conditioned on not drawing $\perp$. We will consider families of bidirectional channels $\{\mathcal{B}_k\}_{k\geq 0}$ such that: (1) the length of elements in D_k is polynomially-bounded in k; (2) for each $h \in \mathcal{H}_k$ and party P, either $\Pr[\mathcal{B}_h^P = \perp] = 1$ or $\Pr[\mathcal{B}_h^P = \perp] \leq 1 - \delta$, for constant δ; and (3) there exists a function $\ell(k) = \omega(\log k)$ so that for each $h \in \mathcal{H}_k$, $H_\infty((\mathcal{B}_h^P)_k \neq\perp) \geq \ell(k)$ (that is, there is some variability in the communications).

We assume that party P can draw from $\mathcal{B}_h^P$ for any history h, and that the adversary can draw from $\mathcal{B}_h^P$ for every party P and history h. We assume that the ability to draw from these distributions does not contradict the cryptographic assumptions that our results are based on. In the rest of the paper, all communications will be assumed to conform to the bidirectional channel structure: parties only communicate by sending documents from D to each other and parties not running a protocol communicate according to the distributions specified by $\mathcal{B}$. Parties running a protocol strive to communicate using sequences of documents that appear to come from $\mathcal{B}$. As a convention, when $\mathcal{B}$ is compared to another random variable, we mean a random variable which draws from the process $\mathcal{B}$ the same number of documents as the variable we are comparing it to.

3. COVERT TWO-PARTY COMPUTATION WITH SEMI-HONEST ADVERSARIES

We now present a protocol for covert two-party computation that is secure against semi-honest adversaries in the standard model (without Random Oracles) and assumes that the decisional Diffie-Hellman problem is hard. The protocol is based on Yao's well-known function evaluation protocol [22].

We first define covert two-party computation formally, and we then describe Yao's protocol and the necessary modifications to turn it into a covert computation protocol. The definition presented in this section is only against honest-but-curious adversaries and is unfair in that only one of the parties obtains the result. In Section 4 we will define covert two-party computation against malicious adversaries and present a protocol that is fair: either both parties obtain the result at roughly the same time or neither of them does. The protocol in Section 4 uses the honest-but-curious protocol presented in this section as a subroutine.

3.1 Definitions

Formally, a two-party, n-round protocol is a pair $\Pi = (P_0, P_1)$ of programs. The computation of Π proceeds as follows: at each round, P_0 is run on its input x_0, the security parameter 1^k, a state s_0, and the (initially empty) history of messages exchanged so far, to produce a new message m and an internal state s_0. The message m is sent to P_1, which is run on its input x_1, the security parameter 1^k, a state s_1, and the history of messages exchanged so far to produce a message that is sent back to P_0, and a state s_1 to be used in the next round. Denote by $\langle P_0(x_0), P_1(x_1)\rangle$ the *transcript* of the interaction of P_0 with input x_0 and P_1 with input x_1. This transcript includes all messages exchanged between P_0 and P_1 along with the timestep in which they were sent. After n rounds, each party $P \in \{P_0, P_1\}$ halts with an output, denoted by $\Pi_P(x_0, x_1) = \Pi_P(\bar{x})$. We say that Π *correctly realizes the functionality* f if for at least one $P \in \{P_0, P_1\}$, $\Pr[\Pi_P(\bar{x}) = f(\bar{x})] \geq 1 - \nu(k)$, where ν is negligible.

For $\sigma \in \{0, 1\}$, we denote by $V_\Pi^{P_\sigma}(x_0, x_1)$ the *view* of party P_σ on input x_σ when interacting with $P_{1-\sigma}$ on input $x_{1-\sigma}$. The view includes P_σ's input x_σ, private random bits, and all messages sent by P_0 and P_1. We say Π *securely realizes the functionality* f if Π correctly realizes f and, for any P'_σ and $x_{1-\sigma}$, there is a *simulator* P''_σ and an x_σ such that $P''_\sigma(f(x_0, x_1)) \approx V_\Pi^{P'_\sigma}(x_0, x_1)$. Notice that given $f(x_0, x_1)$, P'_σ could just use P''_σ to simulate his interaction with $P_{1-\sigma}$ without actually running Π. Thus if Π securely implements f, neither party learns more from the interaction than could be learned from just $f(x_0, x_1)$.

Define the view of party P interacting in protocol Π up through round j by $V_{\Pi,j}^P(\bar{x})$. When party P_σ is not executing Π but is drawing from $\mathcal{B}$ instead, we denote this "protocol" by $\Pi : \mathcal{B}_\sigma$.

Definition 1. *(Covert two-party protocol against honest-but-curious adversaries) We say an n-round, two-party protocol* (P_0, P_1) covertly realizes the functionality f for bidirectional channel $\mathcal{B}$ *if it securely realizes f and if it has the following additional properties:*

1. *(External covertness): For any input $\bar{x}$,*
$$\langle P_0(x_0), P_1(x_1)\rangle \approx \mathcal{B} \ .$$
2. *(Internal covertness): For any input $\bar{x}$, $V_{\Pi,n}^{P_0}(\bar{x}) \approx V_{\Pi:\mathcal{B}_1,n}^{P_0}(\bar{x})$ and $V_{\Pi,n-1}^{P_1}(\bar{x}) \approx V_{\Pi:\mathcal{B}_0,n-1}^{P_1}(\bar{x})$.*
3. *(Final Covertness): For every PPT D there exists a PPT D' and a negligible ν such that for any x_1 and any distribution X_0, $\mathbf{Adv}^D_{V_\Pi^{P_1}(X_0,x_1),V_{\Pi:\mathcal{B}_0}^{P_1}(X_0,x_1)}(k) \leq \mathbf{Adv}^{D'}_{f(X_0,x_1),U_l}(k) + \nu(k)$.*

In other words, until the final round, neither party can distinguish between the case that the other is running the protocol or just drawing from $\mathcal{B}$; and after the final message, P_0 still cannot tell, while P_1 can only distinguish the cases if $f(x_0, x_1)$ and U_m are distinguishable.

We will slightly abuse notation and say that a protocol which has messages indistinguishable from random bits (even given one party's view) is *covert for the uniform channel $\mathcal{U}$*.

3.2 Modifying Yao's Protocol For Two-Party Secure Function Evaluation

Yao's protocol [22] securely (not covertly) realizes any functionality f that is expressed as a combinatorial circuit. The protocol is run between two parties, the *Input Owner* A and the *Program Owner* B. The input of A is a value x, and the input of B is a circuit to compute the function f. At the end of the protocol, A learns $f(x)$ (and nothing else about f) and B learns nothing about x. At a high level view, the protocol and its modification work as follows (we describe all the details in the Appendix):

- B first produces a "garbled" version of the circuit by assigning to every wire of the circuit two random k-bit strings to denote the values 0 and 1.
- B then uses a semantically secure encryption scheme to produce a description of the circuit such that given the garbled input values corresponding to x, A can compute the garbled output.
- A and B then perform $|x|$ executions of 1-out-of-2 oblivious transfer (OT_1^2) in which A learns the garbled input values corresponding to x and nothing else. In the Appendix, we show how to modify a OT_1^2 protocol from [19] so that it is covert for the uniform channel:

PROTOCOL 1. *(Basic steganographic encoding/decoding routines)*

Procedure $\texttt{Encode}^{\mathcal{D}}$:	Procedure $\texttt{Decode}$:
Input: $H \in \mathbb{H}$, $c \in \{0,1\}^m$	**Input:** $H \in \mathbb{H}$, $s \in D$
Let $j = 0$	set $c = H(s)$
repeat:	**Output:** c
sample $s \leftarrow \mathcal{D}$, set $j = j + 1$	
until $H(s) = c$ OR $(j > k)$	
Output: s	

LEMMA 1. *Under the Decisional Diffie-Hellman assumption, there exists a protocol which covertly realizes OT_1^2 for the uniform channel.*

- Finally, B sends A the encrypted description of the circuit and an interpretation table that allows A to interpret the garbled output values.

LEMMA 2. *It is possible to instantiate Yao's protocol so that the garbled circuit and interpretation table are indistinguishable from uniform bits*

For completeness, we describe the full protocol in the Appendix, where we also prove:

THEOREM 1. *Any functionality f can be covertly realized for the uniform channel, $\mathcal{U}$.*

3.3 Steganographic Encoding

We use provably secure steganography to transform the above protocol into a covert two-party protocol for any bidirectional channel $\mathcal{B}$ satisfying the properties mentioned in Section 2. We also use steganography as a building block for all other covert protocols presented in this paper. For completeness we state a construction that has appeared in various forms in [7, 17, 2, 3].

Let $\mathbb{H}$ denote a family of hash functions $H : D \to \{0,1\}^c$ which is *pairwise independent*. Let $\mathcal{D}$ denote an arbitrary probability distribution on D. The functions in Protocol 1 hide and recover m uniformly-chosen bits in a distribution indistinguishable from $\mathcal{D}$ when $H_\infty(\mathcal{D}) = \omega(\log k)$ and $m = O(\log k)$.

PROPOSITION 1. *Let $H \leftarrow \mathbb{H}$. Then*

$$\Delta\left((H, \texttt{Encode}^{\mathcal{D}}(H, U_m)), (H, \mathcal{D})\right) \le 2^{-(\ell(k)-m)/2+1} .$$

The result follows from the Leftover Hash Lemma ([16], Lemma 4.8). Intuitively, it guarantees that $\texttt{Encode}(c)$ will be (statistically) indistinguishable from the messages exchanged in a bidirectional channel whenever c is a uniformly chosen bit string. (When we refer to $\texttt{Encode}$ with only a single argument, we implicitly assume that an appropriate H has been chosen and is publicly accessible to all parties.)

Thus, to guarantee covertness for channel $\mathcal{B}$, we will ensure that all our protocols generate messages that are indistinguishable from uniformly chosen random bits and then encode these messages with $\texttt{Encode}$. Formally, suppose $\Pi = (P_0, P_1)$ is an arbitrary two-party protocol which securely realizes the functionality f. We will construct a protocol $\Sigma^\Pi = (S_0^{P_0}, S_1^{P_1})$ which has the property that if $V_\Pi^{P_b}(\bar{x})$ is indistinguishable from uniformly chosen bits (that is, Π covertly realizes f for the uniform channel), then Σ^Π covertly realizes the functionality f for channel $\mathcal{B}$. We assume that P_0, P_1 have the property that, given a partial input, they return the string ε, indicating that more bits of input are needed. Then $S_b^{P_b}$ has the following round function (which simply uses $\texttt{Encode}$ and $\texttt{Decode}$ to encode and decode all messages exchanged by P_0 and P_1):

PROTOCOL 2. *(Transformation to a covert protocol)*

Procedure $S_b^{P_b}$:
Input: history $h \in \mathcal{H}$, $\texttt{state}$, document $s \in D$
draw $d \leftarrow \mathcal{B}_h^{P_b}$
if ($\texttt{state}$.status = "receiving") then
  set $\texttt{state}.msg = \texttt{state}.msg \| \texttt{Decode}(s)$;
  set $c = P_b(\texttt{state}.msg)$
  if ($c \neq \varepsilon$) set $\texttt{state}$.status = "sending"; set $\texttt{state}.msg = c$
if ($\texttt{state}$.status = "sending" and $d \neq \perp$) then
  set $c, \texttt{state}.msg = \texttt{state}.msg$, where $|c| = m$;
  set $d = \texttt{Encode}^{(\mathcal{B}_h^{P_b} \neq \perp)}(c)$
  if $\texttt{state}.msg$ = "" set $\texttt{state}$.status = "receiving"
Output: message d, $\texttt{state}$

THEOREM 2. *If Π covertly realizes the functionality f for the uniform channel, then Σ^Π covertly realizes f for the bidirectional channel $\mathcal{B}$.*

PROOF. Let k^c be an upper bound on the number of bits in $\langle P_0(x_0), P_1(x_1)\rangle$. Then Σ^Π transmits at most $2k^c/m$ (non-empty) documents. Suppose there is a distinguisher D for $V_\Sigma^{S_b}(\bar{x})$ from $V_{\Sigma:\mathcal{B}_{1-b}}^{S_b}(\bar{x})$ with significant advantage ϵ. Then D can be used to distinguish $V_\Pi^{P_b}(\bar{x})$ from $V_{\Pi:\mathcal{U}_{1-b}}^{P_b}(\bar{x})$, by simulating each round as in Σ to produce a transcript T; If the input is uniform, then $\Delta(T, \mathcal{B}) \le (k^c/m)2^{-(\ell(k)-m)/2} = \nu(k)$, and if the input is from Π, then T is identical to $V_\Sigma^{S_b}(\bar{x})$. Thus D's advantage in distinguishing Π from $\Pi : \mathcal{U}_{1-b}$ is at least $\epsilon - \nu(k)$. $\square$

IMPORTANT: For the remainder of the paper we will present protocols Π that covertly realize f for $\mathcal{U}$. It is to be understood that the final protocol is meant to be Σ^Π, and that when we state that "Π covertly realizes the functionality f" we are referring to Σ^Π.

3.4 Combining The Pieces

We can combine Lemma 1 along with Theorem 1 to construct a protocol which covertly realizes any two-party functionality. The final protocol, which we call COVERT-YAO, is simple: assume that both parties know a circuit C_f computing the functionality f. Bob first uses the modified Yao's protocol to create a garbled circuit for $f(\cdot, x_B)$. Alice and Bob perform $|x_A|$ modified oblivious transfers for the garbled wire values corresponding to Alice's inputs. Bob sends the garbled gates to Alice, along with the information necessary to de-garble the outputs. Finally, Alice outputs $f(x_A, x_B)$.

THEOREM 3. *The* COVERT-YAO *protocol covertly realizes the functionality f.*

Notice that as the protocol COVERT-YAO is described, it is *necessary* that Bob does not learn the output, because a malicious Bob could give Alice a garbled circuit with different operations in the gates, which could actually output some constant message giving away Alice's participation even when the value $f(x_0, x_1)$ would not.

4. FAIR COVERT TWO-PARTY COMPUTATION AGAINST MALICIOUS ADVERSARIES

The protocol presented in the previous section has two serious weaknesses. First, because Yao's construction conceals the function of the circuit, a malicious Bob can garble a circuit that computes some function other than the result Alice agreed to compute. In particular, the new circuit could give away Alice's input or output some distinguished string that allows Bob to determine that Alice is running the protocol. Additionally, the protocol is *unfair:* either Alice or Bob does not get the result.

In this section we present a protocol that avoids these problems. In particular, our solution has the following properties: (1) If both parties follow the protocol, both get the result; (2) If Bob cheats by garbling an incorrect circuit, neither party can tell whether the other is running the protocol, except with negligible advantage; and (3) Except with negligible probability, if one party terminates early and computes the result in time T, the other party can compute the result in time at most $O(T)$. Our protocol is secure in the random oracle model, under the Decisional Diffie-Hellman assumption. We show at the end of this section, however, that our protocol can be made to satisfy a slightly weaker security condition without the use of a random oracle. (We note that the technique used in this section has some similarities to one that appears in [1].)

4.1 Definitions

We assume the existence of a non-interactive bitwise commitment scheme with commitments that are indistinguishable from random bits. An example is the well-known scheme that commits to bit b by $cmt(b;(r,x)) = r\|\pi(x)\|(x\cdot r)\oplus b$, where π is a one-way permutation on domain $\{0,1\}^k$, $x\cdot y$ denotes the inner-product of x and y over $GF(2)$, and $x,r \leftarrow U_k$. (The integer DDH assumption implies the existence of π.) Commitment to a multiple-bit string s, $CMT(s;\cdot)$, can be implemented by committing to the individiual bits of s.

Let f denote the functionality we wish to compute. We say that f is *fair* if for every distinguisher D_σ distinguishing $f(X_0,X_1)$ from U given X_σ with advantage at least ϵ, there is a distinguisher $D_{1-\sigma}$ with advantage at least $\epsilon - \nu(k)$, for a negligible function ν. (That is, if P_0 can distinguish $f(X_0,X_1)$ from uniform, so can P_1.) We say f is *strongly fair* if $(f(X_0,X_1),X_0) \approx (f(X_0,X_1),X_1)$.

A n-round, two-party protocol $\Pi = (P_0,P_1)$ to compute functionality f is said to be a strongly fair covert protocol for the bidirectional channel $\mathcal{B}$ if the following conditions hold:

- (External covertness): For any input $\bar{x}$,
$$\langle P_0(x_0), P_1(x_1)\rangle \approx \mathcal{B}$$
.
- (Strong Internal Covertness): There exists a PPT E (an *extractor*) such that if PPT $D(V)$ distinguishes between $V^{P_\sigma}_{\Pi,i}(\bar{x})$ and $V^{P_\sigma}_{\Pi:\mathcal{B}_{1-\sigma},i}(\bar{x})$ with advantage ϵ, $E^D(V^{P_\sigma}_{\Pi}(\bar{x}))$ computes $f(\bar{x})$ with probability at least $\epsilon/poly(k)$
- (Strong Fairness): If the functionality f is fair, then for any C_σ running in time T such that $\Pr[C_\sigma(V^{\sigma}_{\Pi,i}(\bar{x})) = f(\bar{x})] \geq \epsilon$, there exists a $C_{1-\sigma}$ running in time $O(T)$ such that
$$\Pr[C_{1-\sigma}(V^{1-\sigma}_{\Pi,i}(\bar{x})) = f(\bar{x})] = \Omega(\epsilon)\ .$$
- (Final Covertness): For every PPT D there exists a PPT D' and a negligible ν such that for any x_σ and distribution $X_{1-\sigma}$, $\mathbf{Adv}^{D}_{V^{P_\sigma}_{\Pi}(X_{1-\sigma},x_\sigma),V^{P_\sigma}_{\Pi:\mathcal{B}_{1-\sigma}}(X_{1-\sigma},x_\sigma)}(k)$
$\leq \mathbf{Adv}^{D'}_{f(X_{1-\sigma},x_\sigma),U_l}(k) + \nu(k)$.

Intuitively, the Internal Covertness requirement states that "Alice can't tell if Bob is running the protocol until she gets the answer," while Strong Fairness requires that "Alice can't get the answer unless Bob can." Combined, these requirements imply that neither party has an advantage over the other in predicting whether the other is running the protocol.

4.2 Construction

As before, we have two parties, P_0 (Alice) and P_1 (Bob), with inputs x_0 and x_1, respectively, and the function Alice and Bob wish to compute is $f : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \rightarrow \{0,1\}^l$, presented by the circuit C_f. The protocol proceeds in three stages: COMMIT, COMPUTE, and REVEAL. In the COMMIT stage, Alice picks $k+2$ strings, r_0, and $s_0[0]$, $s_0[1]$, …, $s_0[k]$, each k bits in length. Alice computes commitments to these values, using a bitwise commitment scheme which is indistinguishable from random bits, and sends the commitments to Bob. Bob does likewise (picking strings $r_1, s_1[0], \ldots, s_1[k]$).

The next two stages involve the use of a pseudorandom generator $G : \{0,1\}^k \rightarrow \{0,1\}^l$ which we will model as a random oracle *for the security argument only*: G itself must have an efficiently computable circuit. In the COMPUTE stage, Alice and Bob compute two serial runs ("rounds") of the covert Yao protocol described in the previous section. If neither party cheats, then at the conclusion of the COMPUTE stage, Alice knows $f(x_0,x_1)\oplus G(r_1)$ and Bob's value $s_1[0]$; while Bob knows $f(x_0,x_1)\oplus G(r_0)$ and Alice's value $s_0[0]$. The REVEAL stage consists of k rounds, where each round consists of two runs of the COVERT-YAO protocol. At the end of each round i, if nobody cheats, Alice learns the i^{th} bit of Bob's string r_1, labeled $r_1[i]$ and also Bob's value $s_1[i]$. After k rounds in which neither party cheats, Alice thus knows r_1 and can compute $f(x_0,x_1)$ by computing the exclusive-or of $G(r_1)$ with the value she learned in the COMPUTE stage, and Bob can likewise compute the result.

Each circuit sent by Alice must check that Bob has obeyed the protocol; thus at every round of every stage, the circuit that Alice sends to Bob takes as input the opening of all of Bob's commitments, and checks to see that all of the bits Alice has learned so far are consistent with Bob's input. The difficulty to overcome with this approach is that the result of the check cannot be returned to Alice without giving away that Bob is running the protocol. To solve this problem, Alice's circuits also take as input the last value $s_0[i-1]$ that Bob learned. If Alice's circuit ever finds that the bits she has learned are inconsistent with Bob's input, or that Bob's input for $s_0[i-1]$ is not consistent with the actual value of $s_0[i-1]$, the output is a uniformly chosen string of the appropriate length. Once this happens, all future outputs to Bob will also be independently and uniformly chosen, because he will have the wrong value for $s_0[i]$, which will give him the wrong value for $s_0[i+1]$, etc. Thus the values $s_0[1],\ldots,s_0[k]$ serve as "state" bits that Bob maintains for Alice. The analogous statements hold for Bob's circuits and Alice's inputs.

PROTOCOL 3. *(Fair covert two-party computation)*

Inputs and setup. To begin, each party P_σ chooses $k+2$ random strings r_σ, $s_\sigma[0]$,…,$s_\sigma[k] \leftarrow U_k$. P_σ's inputs to the protocol are then $X_\sigma = (x_\sigma, r_\sigma, s_\sigma[0], \ldots, s_\sigma[k])$.

COMMIT stage. Each party P_σ computes the commitment $\kappa_\sigma = CMT(X_\sigma; \rho_\sigma)$ using randomness ρ_σ, and sends this commitment to the other party. Denote by K_σ the value that P_σ interprets as a commitment to $X_{1-\sigma}$, that is, K_0 denotes the value Alice interprets as a commitment to Bob's input X_1.

COMPUTE stage. The COMPUTE stage consists of two serial runs of the COVERT-YAO protocol.

1. Bob garbles the circuit COMPUTE_1 shown in figure 1, which takes x_0, r_0, $s_0[0]$,...,$s_0[k]$, and ρ_0 as input and outputs $G(r_1) \oplus f(x_0, x_1) \| s_1[0]$ if K_1 is a commitment to X_0. If this check fails, COMPUTE_1 outputs a uniformly chosen string, which has no information about $f(x_0, x_1)$ or $s_1[0]$. Bob and Alice perform the COVERT-YAO protocol; Alice labels her result $F_0 \| S_0[0]$.

2. Alice garbles the circuit COMPUTE_0 shown in figure 1, which takes x_1, r_1, $s_1[0]$,...,$s_1[k]$, and ρ_1 as input and outputs $G(r_0) \oplus f(x_0, x_1) \| s_0[0]$ if K_0 is a commitment to X_1. If this check fails, COMPUTE_0 outputs a uniformly chosen string, which has no information about $f(x_0, x_1)$ or $s_0[0]$. Bob and Alice perform the COVERT-YAO protocol; Bob labels his result $F_1 \| S_1[0]$.

REVEAL stage. The REVEAL stage consists of k rounds, each of which consists of 2 runs of the COVERT-YAO protocol:

1. In round i, Bob garbles the circuit REVEAL_1^i shown in figure 1, which takes input $x_0, S_0[i-1], r_0, s_0[0 \dots k], \rho_0$ and checks that:

   - Bob's result from the COMPUTE stage, F_1, is consistent with x_0, r_0.
   - The bit $R_1[i-1]$ which Bob learned in round $i-1$ is equal to bit $i-1$ of Alice's secret r_0. (By convention, and for notational uniformity, we will define $R_0[0] = R_1[0] = r_0[0] = r_1[0] = 0$)
   - The state $S_0[i-1]$ that Bob's circuit gave Alice in the previous round was correct. (Meaning Alice obeyed the protocol up to round $i-1$.)
   - Finally, that the state $S_1[i-1]$ revealed to Bob in the previous round was the state $s_0[i-1]$ which Alice committed to in the COMMIT stage.

   If all of these checks succeed, Bob's circuit outputs bit i of r_1 and state $s_1[i]$; otherwise the circuit outputs a uniformly chosen $(k+1)$-bit string. Alice and Bob perform COVERT-YAO and Alice labels the result $R_0[i], S_0[i]$.

2. Alice garbles the circuit REVEAL_0^i depicted in figure 1 which performs the analogous computations to REVEAL_1^i, and performs the COVERT-YAO protocol with Bob. Bob labels the result $R_1[i], S_1[i]$.

After k such rounds, if Alice and Bob have been following the protocol, we have $R_1 = r_0$ and $R_0 = r_1$ and both parties can compute the result. The "states" s are what allow Alice and Bob to check that all previous outputs and key bits (bits of r_0 and r_1) sent by the other party have been correct, without ever receiving the results of the checks or revealing that the checks fail or succeed.

THEOREM 4. *Construction 3 is a strongly fair covert protocol realizing the functionality f*

PROOF. (Sketch) The correctness of the protocol follows by inspection. The two-party security follows by the security of Yao's protocol. Now suppose that some party, wlog Alice, cheats (by sending a circuit which computes an incorrect result) in round j. Then, the key bit $R_0[j+1]$ and state $S_0[j+1]$ Alice computes in round $j+1$ will be randomized; and with overwhelming probability every subsequent result that Alice computes will be useless. Assuming Alice can distinguish $f(x_0, X_1)$ from uniform, she can still compute the result in at most 2^{k-j} time by exhaustive search over the remaining key bits. By successively guessing the round at which Alice began to cheat, Bob can compute the result in time at most 2^{k-j+2}. If Alice aborts at round j, Bob again can compute the result in time at most 2^{k-j+1}. If Bob cheats in round j by giving inconsistent inputs, with high probability all of his remaining outputs are randomized; thus cheating in this way gives him no advantage over aborting in round $j-1$. Thus, the fairness property is satisfied.

If G is a random oracle, neither Alice nor Bob can distinguish anything in their view from uniformly chosen bits without querying G at the random string chosen by the other. So given a distinguisher D running in time $p(k)$ for $V_{\Pi,i}^{P_0}(\bar{x})$ with advantage ϵ, it is simple to write an extractor which runs D, recording its queries to G, picks one such query (say, q) uniformly, and outputs $G(q) \oplus F_0$. Since D can only have an advantage when it queries r_1, E will pick $q = r_1$ with probability at least $1/p(k)$ and in this case correctly outputs $f(x_0, x_1)$. Thus the Strong Internal Covertness property is satisfied. □

4.3 Weakly Fair Covertness

We can achieve a slightly weaker version of covertness without using random oracles. Π is said to be a *weakly fair* covert protocol for the channel $\mathcal{B}$ if Π is externally covert, and has the property that if f is strongly fair, then for every distinguisher D_σ for $V_{\Pi,i}^{P_\sigma}(\bar{x})$ with significant advantage ϵ, there is a distinguisher $D_{1-\sigma}$ for $V_{\Pi,i}^{P_{1-\sigma}}(\bar{x})$ with advantage $\Omega(\epsilon)$. Thus in a weakly fair covert protocol, we do not guarantee that both parties get the result, only that if at some point in the protocol, one party can tell that the other is running the protocol with significant advantage, the same is true for the other party.

We note that in the above protocols, if the function G is assumed to be a pseudorandom generator (rather than a random oracle), then the resulting protocol exhibits weakly fair covertness. Suppose D_σ has significant advantage ϵ after round $i = 2j$, as in the hypothesis of weak covertness. Notice that given $r_{1-\sigma}[1], \dots, r_{1-\sigma}[j{-}1], G(r_{1-\sigma}) \oplus f(\bar{x})$, the remainder of P_σ's view can be simulated efficiently. Then D_σ must be a distinguisher for $G(r)$ given the first $j-1$ bits of r. But since f is strongly fair, $P_{1-\sigma}$ can apply D_σ to $G(r_\sigma) \oplus f(\bar{x})$ by guessing at most 1 bit of r_σ and simulating P_σ's view with his own inputs. Thus $P_{1-\sigma}$ has advantage at least $\epsilon/2 - \nu(k) = \Omega(\epsilon)$.

5. CONCLUSION AND OPEN QUESTIONS

We have presented protocols for covert two-party computation that combine steganography with cryptographic protocols whose messages are all indistinguishable from uniformly chosen random bits. Covert two-party computation can be applied whenever the use of ordinary two-party computation raises enough suspicion to defeat its intended purpose. Our protocols are secure against semi-honest adversaries under the decisional Diffie-Hellman assumption and against malicious adversaries in the Random Oracle model.

Our work leaves room for improvement and open problems. For example, given the known theoretical issues with

COMPUTE$_\sigma(x_{1-\sigma}, r, s[0\ldots k], \rho) =$
if $(K_\sigma = CMT(x_{1-\sigma}, r, s; \rho))$
then
  set $F = G(r_\sigma) \oplus f(x_0, x_1)$
  set $S = s_\sigma[0]$
else
  draw $F \leftarrow U_l, S \leftarrow U_k$.
output $F \| S$

REVEAL$^i_\sigma(x_{1-\sigma}, S_{1-\sigma}[i-1], r, s_{1-\sigma}[0\ldots k], \rho) =$
Let $F = G(r) \oplus f(x_0, x_1)$
if $(K_\sigma = CMT(x_{1-\sigma}, r, s_{1-\sigma}; \rho)$
  and $F = F_\sigma$
  and $R_\sigma[i-1] = r[i-1]$
  and $S_{1-\sigma}[i-1] = s_\sigma[i-1]$
  and $S_\sigma[i-1] = s_{1-\sigma}[i-1])$
then
  set $R = r_\sigma[i], S = s_\sigma[i]$
else
  draw $R \leftarrow \{0,1\}$, $S \leftarrow U_k$
output $R \| S$

Figure 1: The circuits COMPUTE and REVEAL.

the random oracle methodology [8, 10, 15, 4], it is an important open problem to remove the RO assumption from the security proof for Construction 3. It seems at least plausible that constructions based on concrete assumptions such as the "knowledge-of-exponent" assumption or the "generalized BBS" assumption may allow construction of standard fair covert protocols, yet the obvious applications always destroy the final covertness property. Another open question is that of improving the efficiency of the protocols presented here, either by designing protocols for specific goals or through adapting efficient two-party protocols to provide covertness.

An interesting question is whether the notion of covert two-party computation can be extended in some natural and implementable way to multiple parties. Such a generalization could have important applications in the area of anonymous communication, allowing, for instance, the deployment of undetectable anonymous remailer networks.

Shout outs to... Our homeboy Manuel Blum, our peeps Avrim Blum, Shuchi Chawla, Dr. Evil, Adam Tauman Kalai, Yael Tauman Kalai, Benny Pinkas, Mike Reiter, David Wagner, and several anonymous reviewers for their helpful comments on earlier versions of this work; the National Science Foundation for Grants CCR-0122581 and CCR-0058982 (The Aladdin Center), as well as several agencies that have supported our work on bribery but wish to remain anonymous. Luis von Ahn was partially supported by a Microsoft Research fellowship — Thanks Bill!

6. REFERENCES

[1] G. Aggarwal, N. Mishra and B. Pinkas. Secure computation of the k'th-ranked element. In: *Advances in Cryptology – Proceedings of Eurocrypt '04*, pages 40–55, 2004.

[2] L. von Ahn and N. Hopper. Public-Key Steganography. In: *Advances in Cryptology – Proceedings of Eurocrypt '04*, pages 323–341, 2004.

[3] M. Backes and C. Cachin. Public-Key Steganography with Active Attacks. in *Theory of Cryptography Conference (TCC)*, pages 210-226, 2005.

[4] M. Bellare, A. Boldyreva, and A. Palacio An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In: *Advances in Cryptology — Eurocrypt 2004*, pages 171–188, 2004.

[5] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. *Advances in Cryptology – Proceedings of CRYPTO '89*, pages 547-557, 1990.

[6] E. Brickell, D. Chaum, I. Damgärd, J. van de Graaf: Gradual and Verifiable Release of a Secret. *Advances in Cryptology – Proceedings of CRYPTO '87*, pages 156-166, 1987.

[7] C. Cachin. An Information-Theoretic Model for Steganography. *Information Hiding, 2nd International Workshop*, pages 306-318, 1998.

[8] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, revisited. In: *Thirtieth Annual ACM Symposium on Theory of Computing*, pages 209–218, 1998.

[9] N. Dedic, G. Itkis, L. Reyzin and S. Russell. Upper and Lower Bounds on Black-Box Steganography. To appear in *Theory of Cryptography Conference (TCC)*, 2005.

[10] C. Dwork, M. Naor, O. Reignold, and L. Stockmeyer. Magic Functions. In: *Proceedings of the Fortieth IEEE Symposium on Foundations of Computer Science*, pages 523–534 ,1999

[11] Z. Galil, S. Haber, M. Yung. Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model. *Advances in Cryptology – Proceedings of CRYPTO '87*, pages 135-155, 1987.

[12] O. Goldreich. Secure Multi-Party Computation. Unpublished Manuscript. *http://philby.ucsd.edu/books.html*, 1998.

[13] O. Goldreich, S. Goldwasser and S. Micali. How to construct pseudorandom functions. *Journal of the ACM*, vol 33, 1998.

[14] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. *Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218-229.

[15] S. Goldwasser and Y.T. Kalai. On the (in)security of the Fiat-Shamir paradigm. In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 102–113, 2003.

[16] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4), pages 1364-1396, 1999.

[17] N. Hopper, J. Langford and L. Von Ahn. Provably Secure Steganography. *Advances in Cryptology – Proceedings of CRYPTO '02*, pages 77-92, 2002.

[18] M. Naor. Bit Commitment Using Pseudorandomness. J. Cryptology 4(2): 151-158 (1991)

[19] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In: *Proceedings of the 12th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 448–457. 2001.

[20] M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings, 1999 ACM Conference on Electronic Commerce.*

[21] B. Pinkas. Fair Secure Two-Party Computation. In: *Advances in Cryptology – Eurocrypt '03*, pp 87–105, 2003.

[22] A. C. Yao. Protocols for Secure Computation. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982, pages 160–164.
[23] A. C. Yao. How to Generate and Exchange Secrets. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pages 162–167.

Appendix A: Yao's Protocol and Modifications

Yao's protocol [22] securely (not covertly) realizes any functionality f that is expressed as a combinatorial circuit. Our description is based on [20]. The protocol is run between two parties, the *Input Owner* A and the *Program Owner* B. The input of A is a value x, and the input of B is a description of a function f. At the end of the protocol, B learns $f(x)$ (and nothing else about x), and A learns nothing about f. The protocol requires two cryptographic primitives, pseudorandom functions and oblivious transfer, which we describe here for completeness.

Pseudorandom Functions. Let $\{F : \{0,1\}^k \times \{0,1\}^{L(k)} \to \{0,1\}^{l(k)}\}_k$ denote a sequence of function families. Let $\mathbf{A}$ be an oracle probabilistic adversary. We define the *prf-advantage of* $\mathbf{A}$ *over* F as

$$\mathbf{Adv}^{\mathrm{prf}}_{F,\mathbf{A}}(k) = |\Pr_K[\mathbf{A}^{F_K(\cdot)}(1^k) = 1] - \Pr_g[\mathbf{A}^g(1^k) = 1]| ,$$

where $K \leftarrow U_k$ and g is a uniformly chosen function from $L(k)$ bits to $l(k)$ bits. Then F is *pseudorandom* if $\mathbf{Adv}^{\mathrm{prf}}_{F,\mathbf{A}}(k)$ is negligible in k for all polynomial-time $\mathbf{A}$. We will write $F_K(\cdot)$ as shorthand for $F_{|K|}(K,\cdot)$ when $|K|$ is known.

Oblivious Transfer. 1-out-of-2 oblivious transfer (OT^2_1) allows two parties, the *sender* who knows the values m_0 and m_1, and the *chooser* whose input is $\sigma \in \{0,1\}$, to communicate in such a way that at the end of the protocol the chooser learns m_σ, while learning nothing about $m_{1-\sigma}$, and the sender learns nothing about σ. Formally, let $\mathcal{O} = (S, C)$ be a pair of interactive PPT programs. We say that $\mathcal{O}$ is *correct* if $\Pr[\mathcal{O}_C((m_0, m_1), \sigma) = m_\sigma] \geq 1 - \epsilon(k)$ for negligible ϵ. We say that $\mathcal{O}$ has *chooser privacy* if for any PPT S' and any m_0, m_1,

$$\left| \Pr[S'(\langle S'(m_0, m_1), C(\sigma)\rangle) = \sigma] - \frac{1}{2} \right| \leq \epsilon(k)$$

and $\mathcal{O}$ has *sender privacy* if for any PPT C' there exists a σ and a PPT C'' such that $C''(m_\sigma) \approx V^{C'}_{\Pi}((m_0, m_1), \sigma)$. We say that $\mathcal{O}$ *securely realizes the functionality* OT^2_1 if $\mathcal{O}$ is correct and has chooser and sender privacy.

Yao's Protocol. Yao's protocol is based on expressing f as a combinatorial circuit. Starting with the circuit, the program owner B assigns to each wire i two random k-bit values (W^0_i, W^1_i) corresponding to the 0 and 1 values of the wire. It also assigns a random permutation π_i over $\{0,1\}$ to the wire. If a wire has value b_i we say it has "garbled" value $(W^{b_i}_i, \pi_i(b_i))$. To each gate g, B assigns a unique identifier I_g and a table T_g which enables computation of the garbled output of the gate given the garbled inputs. Given the garbled inputs to g, T_g does not disclose any information about the garbled output of g for any other inputs, nor does it reveal the actual values of the input bits or the output bit.

Assume g has two input wires (i, j) and one output wire *out* (gates with higher fan in or fan out can be accommodated with straightforward modifications). The construction of T_g uses a pseudorandom function F whose output length is $k + 1$. The table T_g is shown in figure 2.

$\pi_i(b_i)$	$\pi_j(b_j)$	value
0	0	$(W^{g(b_i,b_j)}_{out}, \pi_{out}(b_{out})) \oplus F_{W_j^{b_j}}(I_g, 0) \oplus F_{W_i^{b_i}}(I_g, 0)$
0	1	$(W^{g(b_i,b_j)}_{out}, \pi_{out}(b_{out})) \oplus F_{W_j^{b_j}}(I_g, 0) \oplus F_{W_i^{b_i}}(I_g, 1)$
1	0	$(W^{g(b_i,b_j)}_{out}, \pi_{out}(b_{out})) \oplus F_{W_j^{b_j}}(I_g, 1) \oplus F_{W_i^{b_i}}(I_g, 0)$
1	1	$(W^{g(b_i,b_j)}_{out}, \pi_{out}(b_{out})) \oplus F_{W_j^{b_j}}(I_g, 1) \oplus F_{W_i^{b_i}}(I_g, 1)$

Figure 2: Table produced by Yao's protocol for garbled gate g.

To compute $f(x)$, B computes garbled tables T_g for each gate g, and sends the tables to A. Then, for each circuit input wire i, A and B perform an oblivious transfer, where A plays the role of the chooser (with $\sigma = b_i$) and B plays the role of the sender, with $m_0 = W^0_i \| \pi_i(0)$ and $m_1 = W^1_i \| \pi_i(1)$. A computes $\pi_j(b_j)$ for each output wire j of the circuit (by trickling down the garbled inputs using the garbled tables) and sends these values to B, who applies π_j^{-1} to learn b_j. Alternatively, B can send the values π_j (for each circuit output wire j) to A, who then learns the result. Notice that the first two columns of T_g can be implicitly represented, leaving a "table" which is indistinguishable from uniformly chosen bits.

Covert Oblivious Transfer. As mentioned above, we guarantee the security of our protocols by ensuring that all the messages exchanged are indistinguishable from uniformly chosen random bits. To this effect, we present a modification of the protocol in [19] for oblivious transfer that ensures that all messages exchanged are indistinguishable from uniform when the input messages m_0 and m_1 are uniformly chosen. Our protocol relies on the well-known integer decisional Diffie-Hellman assumption:

Integer Decisional Diffie-Hellman. Let P and Q be primes such that Q divides $P-1$, let $\mathbb{Z}^*_P$ be the multiplicative group of integers modulo P, and let $g \in \mathbb{Z}^*_P$ have order Q. Let $\mathbf{A}$ be an adversary that takes as input three elements of $\mathbb{Z}^*_P$ and outputs a single bit. Define the *DDH advantage of* $\mathbf{A}$ *against* $\mathbb{G} = (g, P, Q)$ as:

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathbf{A}}(\mathbb{G}) = |\Pr_{a,b}[\mathbf{A}(g^a, g^b, g^{ab}) = 1] - \Pr_{a,b,c}[\mathbf{A}(g^a, g^b, g^c) = 1]| ,$$

where the probability is taken over the random tape of $\mathbf{A}$, and the uniform choice of a, b, c from $\mathbb{Z}_Q$, and all the multiplications are over $\mathbb{Z}^*_P$. The Integer Decisional Diffie-Hellman assumption (DDH) for the sequence $\mathbb{G}_k = \{(g_k, P_k, Q_k)\}$ states that for every PPT $\mathbf{A}$, $\mathbf{Adv}^{\mathrm{ddh}}_{\mathbf{A}}(\mathbb{G}_k)$ is negligible in k.

Setup. Let $p = rq + 1$ where $2^k < p < 2^{k+1}$, q is a large prime, and $\gcd(r, q) = 1$; let g generate $\mathbb{Z}^*_p$ and thus $\gamma = g^r$ generates the unique multiplicative subgroup of order q; let $\hat{r}$ be the least integer r such that $r\hat{r} = 1 \bmod q$. Assume $|m_0| = |m_1| < k/2$. Let $H : \{0,1\}^{2k} \times \mathbb{Z}_p \to \{0,1\}^{k/2}$ be a pairwise-independent family of hash functions. Define the randomized mapping $\phi : \langle\gamma\rangle \to \mathbb{Z}^*_p$ by $\phi(h) = h^{\hat{r}} g^{\beta q}$, for a uniformly chosen $\beta \in \mathbb{Z}_r$; notice that $\phi(h)^r = h$ and that for a uniformly chosen $h \in \langle\gamma\rangle$, $\phi(h)$ is a uniformly chosen element of $\mathbb{Z}^*_p$. The following protocol is a simple modification of the Naor-Pinkas 2-round oblivious transfer protocol [19]:

PROTOCOL 4. *COT:*

1. On input $\sigma \in \{0,1\}$, C chooses uniform $a, b \in \mathbb{Z}_q$, sets $c_\sigma = ab \bmod q$ and uniformly chooses $c_{1-\sigma} \in \mathbb{Z}_q$. C sets $x = \gamma^a$, $y = \gamma^b$, $z_0 = \gamma^{c_0}$, $z_1 = \gamma^{c_1}$ and sets $x' = \phi(x), y' = \phi(y), z'_0 = \phi(z_0), z'_1 = \phi(z_1)$. If the most significant bits of all of x', y', z'_0, z'_1 are 0, C sends

the least significant k bits of each to S; otherwise C picks new $a, b, c_{1-\sigma}$ and starts over.

2. The sender recovers x, y, z_0, z_1 by raising to the power r, picks $f_0, f_1 \in H$ and then:
   - S repeatedly chooses uniform $r_0, s_0 \in \mathbb{Z}_q$ and sets $w_0 = x^{s_0}\gamma^{r_0}$, $w_0' = \phi(w_0)$ until he finds a pair with $w_0' \leq 2^k$. He then sets $K_0 = z_0^{s_0} y^{r_0}$.
   - S repeatedly chooses uniform $r_1, s_1 \in \mathbb{Z}_q$ and sets $w_1 = x^{s_1}\gamma^{r_1}$, $w_1' = \phi(w_1)$ until he finds a pair with $w_1' \leq 2^k$. He then sets $K_1 = z_1^{s_1} y^{r_1}$.

   S sends $w_0' \| f_0 \| f_0(K_0) \oplus m_0 \| w_1' \| f_1 \| f_1(K_1) \oplus m_1$
3. C recovers $K_\sigma = (w_\sigma')^{rb}$ and computes m_σ.

LEMMA 3. *S cannot distinguish between the case that C is following the COT protocol and the case that C is drawing from U_k; that is,*

$$V^S_{COT}(m_0, m_1, \sigma) \approx V^S_{COT:\mathcal{U}_C}(m_0, m_1, \sigma).$$

PROOF. Suppose that there exists a distinguisher D with advantage ϵ. Then there exists a DDH adversary A with advantage at least $\epsilon/8 - \nu(k)$ for a negligible ν. A takes as input a triple $(\gamma^a, \gamma^b, \gamma^c)$, picks a random bit σ, sets $z_\sigma = \gamma^c$ and picks a uniform $z_{1-\sigma}' \in \{0,1\}^k$, and computes $x' = \phi(\gamma^a), y' = \phi(\gamma^b), z_\sigma' = \phi(z_\sigma)$; if all three are at most 2^k, then A outputs $D(x', y', z_0', z_1')$, otherwise A outputs 0.

Clearly, when $c \neq ab$,

$$\Pr[A(\gamma^a, \gamma^b, \gamma^c) = 1] \geq \frac{1}{8} \Pr[D(V^S_{\mathrm{COT}:\mathcal{U}_C}) = 1] \ ,$$

since the elements passed by A to D are uniformly chosen and D calls A with probability at least 1/8 (since each of x', y', z_σ' are greater than 2^k with probability at most 1/2). But when $c = ab$, then

$$\Pr[A(\gamma^a, \gamma^b, \gamma^c) = 1] \geq (1/8 - \nu(k)) \Pr[D(V^S_{\mathrm{COT}}) = 1] \ ,$$

since the elements passed by A to D are chosen exactly according to the distribution on C's output specified by *COT*; and since the probability that D is invoked by A is at least 1/8 when $c \neq ab$ it can be at most $\nu(k)$ less when $c = ab$, by the Integer DDH assumption. Thus the DDH advantage of A is at least $\epsilon/8 - \nu(k)$. Since $\epsilon/8$ must be negligible by the DDH assumption, we have that D's advantage must also be negligible. □

LEMMA 4. *When $m_0, m_1 \leftarrow U_{k/2}$, C cannot distinguish between the case that S is following the COT protocol and the case that S is sending uniformly chosen strings. That is,*

$$V^C_{COT}(U_{k/2}, U_{k/2}, \sigma) \approx V^C_{COT:\mathcal{U}_S}(U_{k/2}, U_{k/2}, \sigma) \ .$$

PROOF. The group elements w_0, w_1 are uniformly chosen by S; thus when m_0, m_1 are uniformly chosen, the message sent by S must also be uniformly distributed. □

LEMMA 5. *The COT protocol securely realizes the OT^2_1 functionality.*

PROOF. The protocol described by Pinkas and Naor is identical to the COT protocol, with the exception that ϕ is not applied to the group elements x, y, z_0, z_1, w_0, w_1 and these elements are not rejected if they are greater than 2^k. Suppose an adversarial sender can predict σ with advantage ϵ in COT; then he can be used to predict σ with advantage $\epsilon/16 - \nu(k)$ in the Naor-Pinkas protocol, by applying the map ϕ to the elements x, y, z_0, z_1 and predicting a coin flip if not all are less than 2^k, and otherwise using the sender's prediction against the message that COT would send. Likewise, any bit a chooser can predict about (m_0, m_1) with advantage ϵ in COT, can be predicted with advantage $\epsilon/4$ in the Naor-Pinkas protocol: the Chooser's message can be transformed into elements of $\langle\gamma\rangle$ by taking the components to the power r, and the resulting message of the Naor-Pinkas sender can be transformed by sampling from $w_0' = \phi(w_0), w_1' = \phi(w_1)$ and predicting a coin flip if either is greater than 2^k, but otherwise giving the prediction of the COT chooser on $w_0' \| f_0 \| f_0(K_0) \oplus m_0 \| w_1' \| f_1 \| f_1(K_1) \oplus m_1$. □

THEOREM 5. *Protocol COT covertly realizes the uniform-OT^2_1 functionality*

We can combine the components developed up to this point to make a protocol which covertly realizes any two-party functionality. The final protocol, which we call COVERT-YAO, is simple: assume that both parties know a circuit C_f computing the functionality f. Bob first uses Yao's protocol to create a garbled circuit for $f(\cdot, x_B)$. Alice and Bob perform $|x_A|$ covert oblivious transfers for the garbled wire values corresponding to Alice's inputs. Bob sends the garbled gates to Alice. Finally, Alice collects the garbled output values and sends them to Bob, who de-garbles these values to obtain the output.

THEOREM 1. COVERT-YAO *covertly realizes any functionality f for the uniform channel.*

PROOF. That (Alice, Bob) securely realize the functionality f follows from the security of Yao's protocol. Now consider the distribution of each message sent from Alice to Bob:

- In each execution of COT: each message sent by Alice is uniformly distributed
- Final values: these are masked by the uniformly chosen bits that Bob chose in garbling the output gates. To an observer, they are uniformly distributed.

Thus Bob's view, until the last round, is in fact identically distributed when Alice is running the protocol and when she is drawing from $\mathcal{U}$. Likewise, consider the messages sent by Bob:

- In each execution of COT: because the W_i^b from Yao's protocol are uniformly distributed, Theorem 5 implies that Bob's messages are indistinguishable from uniform strings.
- When sending the garbled circuit, the pseudorandomness of F and the uniform choice of the W_i^b imply that each garbled gate, even given one garbled input pair, is indistinguishable from a random string.

Thus Alice's view after all rounds of the protocol is indistinguishable from her view when Bob draws from $\mathcal{U}$.

If Bob can distinguish between Alice running the protocol and drawing from $\mathcal{B}$ after the final round, then he can also be used to distinguish between $f(X_A, x_B)$ and U_l. The approach is straightforward: given a candidate y, use the simulator from Yao's protocol to generate a view of the "data layer." If $y \leftarrow f(X_A, x_B)$, then, by the security of Yao's protocol, this view is indistinguishable from Bob's view when Alice is running the covert protocol. If $y \leftarrow U_l$, then the simulated view of the final step is distributed identically to Alice drawing from $\mathcal{U}$. Thus Bob's advantage will be preserved, up to a negligible additive term. □