# Covert Two-Party Computation

**Abstract.** We introduce the notion of *covert two-party computation.* Whereas ordinary secure two-party computation only guarantees that the inputs of the individual parties remain secret after the execution of the protocol, covert two-party computation non-trivially employs steganography to yield the following additional guarantees: (A) no outside eavesdropper can determine whether the two parties are performing the computation or simply communicating as they normally do; (B) neither party can tell whether the other is running the protocol or simply communicating as they normally do before they learn $f(x_A, x_B)$; (C) after the protocol concludes, each party can only determine if the other ran the protocol insofar as they can distinguish $f(x_A, x_B)$ from uniformly chosen random bits. Covert two-party computation thus allows the construction of protocols that return $f(x_A, x_B)$ only when it equals a certain value or 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 $f(x_A, x_B)$ does not equal the value of interest.* We present aplications and security definitions for covert two-party computation, and we present protocols that are secure under XXX.

## 1 Introduction

Secure two-party computation allows Alice and Bob to evaluate a function of their secret inputs so that neither learns anything other than the otput of the function. A real-world example that is 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 match without risking embarassment: if Bob is not interested in Alice, for instance, the protocol does not reveal whether Alice is interested in him. So goes the example.

However, though 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 ocurrs 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 guarantees the following (in addition to preserving the secrecy of 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) neither party can tell whether the other is running the protocol or simply communicating as they normally do before they learn $f(x_A, x_B)$; (C) after the protocol concludes, each party can only determine if the other ran the protocol insofar as they can distinguish $f(x_A, x_B)$ from uniformly chosen random bits. By defining a functionality $g(x_A, x_B)$ such that $g(x_A, x_B) = f(x_A, x_B)$ whenever $f(x_A, x_B) = Y$ and $g(x_A, x_B)$ be pseudorandom otherwise, covert two-party computation allows the construction of protocols that return $f(x_A, x_B)$ only when it equals a certain value or interest $Y$ (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 $f(x_A, x_B) \neq Y$.*

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

- **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. In case both parties are romantically interested in each other, it is important for *both* to find out, as otherwise the party who first obtains the result would be able to learn the other's input by always pretending he is romantically interested; there would be no harm for him in doing so since the other would never see the result. However, if the protocol is fair (either both parties obtain the result or neither of them does), parties have a deterrence to lie. *Fair covert two-party computation solves the dating problem.*
- **Bribes.** Deciding whether to bribe an official can be a difficult problem. If the official is corrupt, bribery can be extremely helpful and sometimes necessary. However, if the official abides by the law, attempting to bribe them can have extremely negative consequences. Covert two-party computation allows individuals to approach officials and exchange an innocent-looking sequence of messages with the following properties: (1) if the official is willing to accept bribes and the individual is willing to give them, both of them find out; (2) if at least one of them is not willing to participate in the bribe, neither of them will be able to determine if the other even performed the computation; (3) the official's supervisor, even after seeing the entire sequence of messages exchanged, would not be able to determine if the parties performed a computation.
- **Cheating in card games.**

Our protocols make use of provably secure steganography [10, 1, 2] to hide the computation in innocent looking communications. The application of steganography, however, is not trivial. Combining steganography with two-party computation in the obvious black-box manner (i.e., forcing all the parties participating in a protocol to communicate steganographically) yields protocols that are undetectable to an outside observer but does not guarantee that the participants will fail to determine if the computation took place. Depending on the output of the function, we wish to hide that the computation took place even from the participants themselves.

In this paper we introduce security definitions for covert two-party computation. We construct a protocol for covert two-party computation that is fair and secure against mallicious adversaries. The security of our protocol is based on XXX.

## Related Work

To the best of our knowledge, the notion of covert two-party computation is new. The idea of combining steganography with other primitives has been considered before (e.g. [8]), but nobody has considered combining it with two-party computation (or with any other primitive to yield a provably secure result). Below we mention the work that we draw ideas from and use throughout the paper.

**Secure Two-Party Computation.** Two-party computation was introduced by Yao [18]. Since then, there have been several papers on two-party secure computation and we refer the reader to a survey by Goldreich [13] for further references. Constructions that yield fairness for two-party computation were intrduced by Yao [**?**], Galil et al. [**?**], Brickell et al. [**?**], Ben-Or et al. [**?**], Beaver and Goldwasser [**?**], and Goldwasser and Levin [**?**], and most recently, Pinkas [**?**]. Our protocols for covert two-party computation are based on Yao's original protocol and draw ideas from [**?**].

**Steganography.** There has been considerable work on digital steganography, but it wasn't until recently that it became part of modern complexity-theoretic cryptography with the work of Hopper et al. [10]. Since then, a few papers have extended the notion to public-key steganography [1, 2].

Our work relies on the formalization of [10, 1, 2] and uses protocols contained therein, but addresses a different problem.

## 2 Preliminaries

### 2.1 Notation

A function $\mu : \mathbb{N} \to [0,1]$ is said to be *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 $|s|$. The concatenation of string $s_1$ and string $s_2$ is denoted by $s_1||s_2$. We assume the existence of efficient, unambiguous *pairing* and *un-pairing* operations, so $(s_1, s_2)$ is not the same as $s_1||s_2$. We let $U_k$ denote the uniform distribution on $k$ bit strings. If $X$ is a finite set, we let $U(X)$ denote the uniform distribution on $X$. If $\mathcal{C}$ is a distribution with finite support $X$, we define the *minimum entropy* of $\mathcal{C}$, $H_\infty(\mathcal{C})$, as

$$H_\infty(\mathcal{C}) = \min_{x \in X} \left\{ \log_2 \frac{1}{\Pr_{\mathcal{C}}[x]} \right\} .$$

The *statistical distance* between two distributions $\mathcal{C}$ and $\mathcal{D}$ with joint support $X$ is defined by

$$\Delta(\mathcal{C}, \mathcal{D}) = \frac{1}{2} \sum_{x \in X} \left| \Pr_{\mathcal{D}}[x] - \Pr_{\mathcal{C}}[x] \right| .$$

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 every probabilistic polynomial-time A, $\mathbf{Adv}^{A}_{\mathcal{C},\mathcal{D}}(k) = |\Pr[A(\mathcal{C}_k) = 1] - \Pr[A(\mathcal{D}_k) = 1]|$ is negligible in $k$.

### 2.2 Standard Cryptographic Notions

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

$$\mathbf{Adv}^{\mathrm{prf}}_{F,\mathbf{A}}(k) = \left| \Pr_{K \leftarrow U(k), r \leftarrow \{0,1\}^*} [\mathbf{A}_r^{F_K(\cdot)} = 1] - \Pr_{g \leftarrow U(L(k), l(k)), r \leftarrow \{0,1\}^*} [\mathbf{A}_r^{g} = 1] \right| ,$$

where $r$ is the string of random bits used by adversary $\mathbf{A}$. 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 sometimes write $F_k(K, \cdot)$ as $F_K(\cdot)$.

**Oblivious Transfer.** 1-out-of-2 oblivious transfer ($\mathrm{OT}^2_1$) 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 $m_\sigma$, while learning nothing about $m_{1-\sigma}$, and the sender learns nothing about $\sigma$. Formally, let $\mathcal{O} = (S, C)$ be a pair of interactive PPT programs. We say that $\mathcal{O}$ is *correct* if $\Pr[C(\sigma, S(m_0, m_1)) = m_\sigma] \geq 1 - \epsilon(k)$ for negligible $\epsilon$. We say that $\mathcal{O}$ has *chooser privacy* if for any PPT $S'$ and any $m_0, m_1$, $\left|\Pr[S'(m_0, m_1, C(\sigma)) = \sigma] - \frac{1}{2}\right| \leq \epsilon(k)$ and $\mathcal{O}$ has *sender privacy* if for any PPT $C'$ there exists a $\sigma$ and a PPT $C''$ such that $C''(m_\sigma) \approx C'(S(m_0, m_1))$. We say that $\mathcal{O}$ *securely realizes the functionality* $\mathrm{OT}^2_1$ if $\mathcal{O}$ is correct and has chooser and sender privacy.

### 2.3 Bidirectional Channels

We seek to hide the communication patterns of a two-party computation protocol in "ordinary" or "innocent-looking" messages. We define ordinary communication patterns in a manner similar to the notion of a *channel* used by [10, 1]. 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 the difference, we call our channel a *bidirectional channel.*

For simplicity, we assume that time proceeds in rounds. Messages are drawn from a set $D$ of *documents.* Each party maintains a *history* $h$, which represents a lexicographically-ordered list of all documents sent to her and the round in which they were received, as well as all documents sent by her and the round in which they were received. Call the set of well-formed histories $\mathcal{H}$. To each party is associated a family of probability distributions $\mathcal{C} = \{\mathcal{C}_h\}_{h\in\mathcal{H}}$ on $D \cup \{\perp\}$. To distinguish between the different parties, we use $\mathcal{C}^{(A)}$ and $h_A$ to denote party $A$'s family of distributions and history, respectively, and $\mathcal{C}^{(B)}$ and $h_B$ to denote party $B$'s.

The communication over a bidirectional channel $\mathcal{B} = (D, \mathcal{H}, \mathcal{C}^{(A)}, \mathcal{C}^{(B)})$ proceeds as follows. At each round, each party, say $A$, receives messages sent to them in the previous round, updates $h_A$ accordingly, and draws a document $d \leftarrow \mathcal{C}^{(A)}_{h_A}$ (the draw could also result in $\perp$, signifying that no action should be taken that round). The document $d$ is then sent to the other party and $h_A$ is updated.

We will consider bidirectional channels $\mathcal{B}(k)$ with the following properties:

- The length of elements in $D(k)$ is polynomially-bounded in $k$.
- There exists a function $\ell(k) = \omega(\log k)$ so that for each $h \in H$, $H_\infty(\mathcal{C}_h) \geq \ell(k)$.

We assume that each party $A$ may draw from $\mathcal{C}^{(A)}$, and in addition we will assume that the adversary may draw from $\mathcal{C}^{(A)}$ and $\mathcal{C}^{(B)}$. We assume that the ability to sample from these distributions does not contradict the cryptographic assumptions that our results are based.

Bidirectional channels model the distribution of communications over a network and are general enough to express almost any form of communication between two parties. We note, however, that the results in this paper do not depend crucially on the precise formalization of the communication distribution and any reasonable formalization would yield similar results.

### 2.4 Steganography

We present some of the results in [10, 1] in a slightly different language. Let $\mathcal{H}$ denote a family of hash functions $h : D \to \{0,1\}^c$ which is *pairwise independent*, that is, for any $x_1 \neq x_2 \in D$, for any $y_1, y_2 \in \{0,1\}^m$, $\Pr_{h\in\mathcal{H}}[h(x_1) = y_1 \wedge h(x_2) = y_2] = 1/2^{2m}$. Let $\mathcal{D}_1, \ldots, \mathcal{D}_l$ denote an arbitrary sequence of probability distributions on $D$ satisfying $H_\infty(\mathcal{D}_i) = \ell(k)$ were $\ell(k) = \omega(\log k)$. The following constructions hide and recover $lm$ uniformly-chosen bits in sequences which are indistinguishable from $(\mathcal{D}_1, \ldots, \mathcal{D}_l)$ (when $\ell(k) - m(k) = \omega(\log k)$) and have appeared in various forms in [4, 10, 2]:

**Construction 1** *(Basic encoding/decoding routines)*

**Procedure** $\texttt{Basic\_Encode}^{\mathcal{D}_1,\dots,\mathcal{D}_l}$:
**Input:** $h \in \mathcal{H}$, $c_1, \dots, c_l \in \{0,1\}^m$
for $i = 1 \dots l$ do
  Let $j = 0$
  repeat:
    sample $s_i \leftarrow \mathcal{D}_i$, increment $j$
  until $h(s_i) = c_i$ OR $(j > k)$
  **Output:** $s_1, s_2, \dots, s_l$

**Procedure** $\texttt{Basic\_Decode}$:
**Input:** $h \in \mathcal{H}$, $s_1, s_2, \dots, s_l \in D$
for $i = 1 \dots l$ do
  set $c_i = h(s_i)$
set $c = c_1 || c_2 || \cdots || c_l$.
**Output:** $c$

**Proposition 1.** *Let $h \leftarrow \mathcal{H}$, $c \leftarrow U_{lm}$, and $d_i \leftarrow \mathcal{D}_i$. Then*

$$\Delta(\langle h, \texttt{Basic\_Encode}^{\mathcal{D}_1,\dots,\mathcal{D}_l}(h,c)\rangle\ , \langle h, d_1, \dots, d_l\rangle) \leq l 2^{-(\ell(k)-m)/2}$$

The result follows from the Leftover Hash Lemma ([9], Lemma 4.8). When we refer to `Basic_Encode` with only a single argument, we implicitly assume that an appropriate $h$ has been chosen and is publicly accessible to all parties.

## 3 Covert Two-Party Computation Secure Against Honest-but-Curious Adversaries

In this section, we define covert two-party computation against hones-but-curious adversaries. We give an implementation that is secure assumiong OT is possible.

### 3.1 Definition

A covert two-party protocol for functionality $f : \{0,1\}^{l_A} \times \{0,1\}^{l_B} \rightarrow \{0,1\}^l$ is a two-party protocol $\Pi = (A, B)$ with the additional property that until the conclusion of the protocol, neither party can distinguish between the case that the other is running $\Pi$ or drawing from $\mathcal{C}$; and no eavesdropper may distinguish between the case that $A$ and $B$ are running $\Pi$ or drawing from $\mathcal{C}$. Define the view of party $P$ interacting with protocol $\Pi$ up through step $j$ by $V^P_{\Pi,j}(\bar{x})$. When a set $Q$ of parties are not executing $\Pi$ but are drawing from $\mathcal{C}$ instead, we denote this "protocol" by $\Pi!Q$.

We say a $m$-round, two-party protocol $(A, B)$ *covertly realizes the functionality* $f$ for channel $\mathcal{C}$ if we have the following conditions:

1. For any input $\bar{x}$,
$$V^A_{\Pi,m}(\bar{x}) \approx V^A_{\Pi!B,m}(\bar{x})$$
2. For any input $\bar{x}$,
$$V^B_{\Pi,m-1}(\bar{x}) \approx V^B_{\Pi!A,m-1}(\bar{x})$$
3. For every PPT $\mathcal{D}$ there exists a PPT $D'$ and a negligible $\nu$ such that for any $x_B$ and any distribution $X_A$,
$$\mathbf{Adv}^{\mathcal{D}}_{V^B_{\Pi}(X_A,x_B),V^B_{\Pi!A}(X_A,x_B)}(k) \leq \mathbf{Adv}^{\mathcal{D}'}_{f(X_A,x_b),U_l}(k) + \nu(k)$$

In other words, until the final message, neither party can distinguish between the case that the other is running the protocol or just drawing from $\mathcal{C}$; and after the final message, $A$ still cannot tell; while $B$ can only distinguish the cases if $f(x_A, x_B)$ and $U_m$ are distinguishable. Note that this definition also implies that $\langle A(x_A), B(x_B)\rangle \approx \mathcal{C}$, since otherwise $A$ could apply the distinguisher to his view (less the random bits).

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

We describe a secure function evaluation protocol for two parties, which is originally due to Yao [18]; our description is taken from [12]. 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 $f$. We apply a modification of this protocol as a key component in covert two-party computation.

The protocol is based on expressing $f$ as a combinatorial circuit. $B$ assigns to each wire $i$ of the circuit two random values $(W_i^0, W_i^1)$ corresponding to the 0 and 1 values of the wire. It also assigns a random permutation $\pi_i$ over $\{0,1\}$ to the wire. If a wire has value $b_i$ we say it has "garbled" value $(W_i^{b_i}, \pi_i(b_i))$. 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 two 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 $(k)$. The construction of $T_g$ uses a pseudorandom function $F$ whose output length is $|W_k^{b_k}|+1$. The table $T_g$ is as follows:

| $\pi_i(b_i)$ | $\pi_j(b_j)$ | value |
|---|---|---|
| 0 | 0 | $(W_k^{g(b_i,b_j)}, \pi_k(b_k)) \oplus F_{W_j^{b_j}}(I_g, 0) \oplus F_{W_i^{b_i}}(I_g, 0)$ |
| 0 | 1 | $(W_k^{g(b_i,b_j)}, \pi_k(b_k)) \oplus F_{W_j^{b_j}}(I_g, 0) \oplus F_{W_i^{b_i}}(I_g, 1)$ |
| 1 | 0 | $(W_k^{g(b_i,b_j)}, \pi_k(b_k)) \oplus F_{W_j^{b_j}}(I_g, 1) \oplus F_{W_i^{b_i}}(I_g, 0)$ |
| 1 | 1 | $(W_k^{g(b_i,b_j)}, \pi_k(b_k)) \oplus F_{W_j^{b_j}}(I_g, 1) \oplus F_{W_i^{b_i}}(I_g, 1)$ |

To compute $f(x)$, $B$ computes garbled tables $t_I$ for each gate $I$, and sends the tables to $A$. Then, for each 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 $m_s = (W_i^s, b_i^s)$. $A$ computes $\pi_j(b_j)$ for each output bit $j$ and sends these values to $B$, who applies $\pi_j^{-1}$ to learn $b_j$.

Formally, a two-party protocol is a pair $\Pi = (A, B)$ of programs. Denote by $\langle A(x_A), B(x_B)\rangle$ the interaction of $A$ on input $x_A$ and $B$ with input $x_B$, and by $V_\Pi^A(x_A, x_B)$ the *view* of party $A$ on input $x_A$ when interacting with $x_B$. The view includes $A$'s private random bits and all messages sent by $A$ and $B$. We say $\Pi$ *securely realizes the functionality* $f$ if for any $A'$ (resp,B'), for any $x_B$ (resp, $x_A$), there is an $A''$ and an $x'_A$ such that $A''(f(x'_A, x_B)) \approx V_\Pi^{A'}(x_A, x_B)$.

### 3.3 Steganographic padding

For ease of exposition, we will present our protocols in the "data layer." To execute these protocols, it is necessary for two parties to steganographically encode (using `Round_Encode`) the messages generated by these protocols. When one party must wait for data transmitted by the other party, he should continue to send draws from $\mathcal{C}_h$ at appropriate times. To prove security, then, it is only necessary to show that all messages generated are indistinguishable from uniform strings of the same length.

### 3.4 Covert Oblivious Transfer

Here we present a modification of the [3] protocol for oblivious transfer, which is indistinguishable from uniformly chosen messages under the integer DDH assumption, when the input messages $m_0$

and $m_1$ are uniformly chosen. Let $p = rq+1$ where $2^k < p < 2^{k+1}$, $q$ is a large prime, and $(r, q) = 1$; let $g$ generate $\mathbb{Z}_p^*$ and thus $\hat{g} = g^r$ generates the unique multiplicative subgroup of order $q$. 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.

**Protocol** COT:

| S | | C |
|---|---|---|
| S: **Input:** $m_0, m_1$<br>choose $c \leftarrow \mathbb{Z}_p$,<br>subject to $c \leq 2^k$. | | C: **Input:** $\sigma$ |
| | $\xrightarrow{c}$ | chooses $k \in \mathbb{Z}_{p-1}$ subject to<br>$h_\sigma = g^k \bmod p, h_{1-\sigma} = c/g^k \bmod p$ ,<br>$h_0 \leq 2^k$. |
| chooses $r_0, r_1 \in \mathbb{Z}_{p-1}$ subject to<br>$g_b = g^{r_b} \leq 2^k$, for $b \in \{0,1\}$.<br>chooses $y_0, y_1 \leftarrow U_{2k}$.<br>sets $C_b = H(y_b, h_b^{r_b r}) \oplus m_b$, $b \in \{0,1\}$. | $\xleftarrow{h_0}$ | |
| | $\xrightarrow{g_0 \| y_0 \| C_0 \| g_1 \| y_1 \| C_1}$ | computes $m_\sigma = H(y_\sigma, g_\sigma^{rk}) \oplus C_\sigma$. |

**Lemma 1.** *$S$ cannot distinguish between the case that $C$ is following the COT protocol and the case that $C$ is drawing from $U_k$; that is,*

$$V_{COT}^S(m_0, m_1, \sigma) \approx V_{COT!C}^S(m_0, m_1.\sigma)$$

*Proof.* This follows because the message sent by $C$ is uniformly chosen.

**Lemma 2.** *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.*

$$V_{COT}^C(m_0, m_1, \sigma) \approx V_{COT!S}^C(m_0, m_1, \sigma)$$

*Proof.* When $m_0, m_1$ are uniformly chosen all of the messages generated by $S$ are uniformly distributed.

**Lemma 3.** *The COT protocol securely realizes the $OT_1^2$ functionality.*

*Proof.* Clearly, $S$ learns nothing about the value $\sigma$, since the output distribution of $C(c, 0)$ and $C(c, 1)$ are identically distributed. Now, suppose without loss of generality, that $\sigma = 0$. We will argue that for any $(m_0, m_1, m_1')$, the messages generated by $S(m_0, m_1, h_0)$ and $S(m_0, m_1', h_0)$ are indistinguishable to $C$, thus proving that $C$ learns nothing about $m_1$. Consider two hybrid COT protocols: COT1 uniformly chooses a value $r_2 \in \mathbb{Z}_q$ and replaces $h_1^{r_1 r}$ with $\hat{g}^{r_2}$, while COT2 replaces the value $H(y_1, h_1^{r_1 r})$ with a uniformly chosen $k/2$-bit string. Clearly, $S_{COT2}(m_0, m_1, h_0)$

and $S_{COT2}(m_0, m'_1, h_0)$ are indistinguishable. But the leftover-hash-lemma implies that $S_{COT2}(x)$ and $S_{COT1}(x)$ are indistinguishable. And the Decisional Diffie-Hellman assumption implies that $S_{COT1}(x)$ and $S(x)$ are indistinguishable, since a distinguisher between $S_{COT1}(x)$ and $S(x)$ with advantage $\epsilon$ can be used to distinguish between $(\hat{g}^a, \hat{g}^b, \hat{g}^{ab})$ and $(\hat{g}^a, \hat{g}^b, \hat{g}^c)$ with advantage at least $\epsilon/4$, using the techniques from section 2. Thus it must be that $S(m_0, m_1, h_0)$ and $S(m_0, m'_1, h_0)$ are indistinguishable.

Conjoining these three lemmas gives the following theorem:

**Theorem 1.** *Protocol $COT$ covertly realizes the uniform-$OT_1^2$ functionality*

### 3.5 Combining the pieces

We can combine these components with the primitives from section 2 to make a protocol which covertly realizes any two-party functionality. The final protocol is simple: Alice and Bob first perform a steganographic key exchange. The resulting key $K$ is used to encrypt all subsequent "data layer" messages between Alice and Bob. Next, Bob 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 uses `Round_Encode` to send the garbled gates to Alice. Finally, Alice collects the garbled output values and uses `Round_Encode` to send them to Bob, who de-garbles these values to obtain the output.

**Theorem 2.** *The protocol described above covertly realizes the functionality $f$.*

*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 the Key Exchange round: this message is uniformly distributed.
- In each execution of COT: each message sent by Alice is again 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 it follows from Proposition 1 that Bob's view, until the last round, is in fact identically distributed when Alice is running the protocol and when she is drawing from $\mathcal{C}^{(Alice)}$. Likewise, consider the messages sent by Bob:

- In the Key Exchange: this message is uniformly distributed.
- In each execution of COT: because the $W_i^b$ are uniformly distributed, Theorem 1 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, is indistinguishable from a random string.

Thus Proposition 1 implies that Alice's view after all rounds of the protocol is indistinguishable from her view when Bob draws from $\mathcal{C}^{(Bob)}$.

If Bob can distinguish between Alice running the protocol and drawing from $\mathcal{C}$ 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", and then encode this transcript using `Round_Encode`. 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 statistically indistinguishable from Alice drawing from $\mathcal{C}$. Thus Bob's advantage will be preserved, up to a negligible additive term.

## 4 Fair Covert Two-party Computation Secure Against Malicious Adversaries

The protocol presented in the previous section has two serious weaknesses:

- 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.
- 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 follwing 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.
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.

### 4.1 Definitions

We assume the existence of a noninteractive bit commitment scheme with commitments which are indistinguishable from random bits. One example is the scheme of Naor, which can be constructed from any pseudorandom generator.

Let $f$ denote the functionality we wish to compute. We say that $f$ is "fair" if for every distinguisher $D_\sigma$ distinguishing $f(X_0, X_1)$ from $U$ given $X_\sigma$ with advantage at least $\epsilon$, there is a distinguisher $D_{1-\sigma}$ with advantage at least $\epsilon - \nu(k)$, for a negligible function $\nu$.

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

- (External Covertness): $\langle P_0(x_0), P_1(x_1)\rangle \approx \mathcal{C}$
- (Internal Covertness): There exists a PPT $E$ such that if PPT $D(V)$ distinguishes between $V^{P_\sigma}_{\Pi,i}(\bar{x})$ and $V^{P_\sigma}_{\Pi!P_{1-\sigma},i}(\bar{x})$ with advantage $\epsilon$, $E^D(V^{P_\sigma}_{\Pi}(\bar{x}))$ computes $f(\bar{x})$ with probability at least $\epsilon/poly(k)$
- (Fairness): If the functionality $f$ is fair, then if $P_\sigma$ can compute $f(x)$ in time $T$, $P_{1-\sigma}$ can compute $f(\bar{x})$ in time at most $O(T)$.

Intuitively, the Internal Covertness requirement states that "Alice can't tell if Bob is running the protocol until she gets the answer," while 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 he is running the protocol.

### 4.2 An exponentially inefficient construction

For expository reasons, we first give a protocol which becomes exponentially inefficient, but clearly illustrates the idea behind our construction. As before, we have two parties, Alice and Bob, with inputs $x_A$ and $x_B$, respectively, and the function Alice and Bob wish to compute is $f$, presented by the circuit $C_f$. The protocol proceeds in $2k+2$ rounds, where each round is a complete execution of the Honest-But-Curious case. At each round, one party creates a circuit which checks to see if the other party has obeyed the protocol. The difficulty with such an approach is that the answer cannot be returned to the party creating the circuit, or else it will give away that the other party is running the protocol. So instead, the parties take turns giving away one bit of information each time, and each party's circuits will take as input some pseudorandom "state" which was output by the last circuit he created, carrying forward the information that the evaluator has not cheated. Let the function $G : \{0,1\}^k \to D_f$ be modeled by a random oracle.

1. Bob chooses $r_B, K_B \leftarrow U_k$, and prepares a circuit $C_1$ that computes $Commit(r_B) \| G(r_B) \oplus f(x_A, x_B) \| F_{K_B}(1)$. Bob computes the garbled circuit $G_1$, which Alice and Bob use to covertly perform Yao's protocol, giving Alice the result $commit \| R_A \| S_1$.
2. Alice chooses a $r_A, K_A \leftarrow U_k$ and prepares a circuit $C_2$ which takes as input the "ungarbling" of $G_1$ and checks that $G_1$ computes $C_1$. If the check succeeds, $C_2$ outputs $Commit(r_A) \| G(r_A) \oplus R_A \| F_{K_A}(2)$, otherwise it outputs a uniformly chosen string. Alice computes the garbled circuit $G_2$ and Alice and Bob covertly perform Yao's protocol, giving Bob the result $commit \| R_B \| S_2$.
3. Bob prepares a circuit $C_3$ which takes as input the ungarbling of $G_2$, and an alleged state $S$. If $G_2$ computes $C_2$ and $S = S_1$, $C_3$ outputs bit 1 of $r_B$ and $S_3 = F_{K_B}(3)$, otherwise it outputs a uniformly chosen string. After Yao's protocol, Alice knows $G_3$, $S_3$, and $b_3$, the first bit of $r_B$.
4. Alice prepares a circuit $C_4$ which takes as input the ungarbling of $G_3$, an alleged state $S$, and the random bits for $Commit$. $C_4$ checks that the bit $b_3$ was properly opened, that $G_3$ computes $C_3$, and that $S = S_2$; if the check succeeds, it outputs the first bit of $r_A$ and $S_4 = F_{K_A}(4)$, and otherwise a uniformly chosen string.

$\vdots$

$i$. Bob prepares a circuit $C_i$ (where $i = 2j+1$) which takes as input the ungarbling of $G_{i-1}$, an alleged state $S$, and the random bits for $Commit$. $C_i$ checks that the bit $b_{i-1}$ (the $j^{\text{th}}$ bit of $r_A$) was properly opened, that $G_{i-1}$ computes $C_{i-1}$ and that $S = S_{i-2}$; if the check succeeds, it outputs the $j^{\text{th}}$ bit of $r_B$ and $S_i = F_{K_B}(i)$, and otherwise a uniformly chosen string.

$\vdots$

After $2k+2$ such rounds, if Alice and Bob have been following the protocol, each knows $r_A$ and $r_B$ and can compute the result. The pseudorandom "states" are what allow Alice and Bob to check that all previous circuits (and key bits) sent by the other party have been correct, without ever receiving the results of the checks or revealing that the checks fail or succeed.

Now suppose that some party, wlog Alice, cheats (by sending an incorrect circuit) in round $i = 2j$. Then by inspection, the key bit and state Alice computes in round $i+1$ will be randomized; and with high probability every subsequent result that Alice computes will be useless. Assuming Alice can distinguish $f(x_A, X_B)$ 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 $i$, Bob again can compute the result in time at most $2^{k-j+1}$. If Bob cheats in round $i$ by giving incorrect inputs, with high probability all of his remaining outputs are randomized; thus cheating in this way gives him no advantage over aborting in round $i-1$. Thus, the fairness property is satisfied.

Neither Alice nor Bob can distinguish anything in her 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 $V^{Alice}_{\Pi,i}(\bar{x})$ with advantage $\epsilon$, it is simple to write an extractor which runs $D$, recording its queries to $G$, picks one such query (say, $q$) uniformly, and outputs $G(q) \oplus R_A$. Since $D$ can only have an advantage when it queries $r_B$, $E$ will pick $q = r_B$ with probability at least $1/p(k)$ and in this case correctly outputs $f(x_A, x_B)$. Thus the Internal Covertness property is satisfied.

Unfortunately, the straightforward implementation of $C_i$ does not scale efficiently. By inspection, the number of gates required to check $G_{i-1}$ is $\Omega(k|C_{i-1}|)$, so that $|C_i| = \Omega(k^i)$. Since the protocol lasts $2k$ rounds, the circuits evaluated in the final rounds will be superpolynomial in length.

### 4.3 Efficient solution

The solution to this problem is to use a more efficient technique for verifying that $G_{i-1}$ computes $C_{i-1}$. In order to do this, we will adapt Micali's *Computationally Sound Proofs* (CS Proofs) [11] to the task. The resulting protocol will yield circuits with size bounded by a polynomial in $k$, giving an efficient protocol for fair covert two-party computation.

**CS Proofs** CS Proofs prove statements of the form $(M, x, y, t)$, where $M$ is a Turing Machine, indicating "$M(x) = y$ in $t$ steps". The length of a proof of $(M, x, y, t)$ as well as the running time of the verifier are bounded by a polynomial in $|M| + |x| + |y| + \log t$. The CS Proof system of [11] works in the random oracle model, and has the property that every true statement can be proven by a CS Proof with probability 1, while false statements can be proven with probability at most $1/2^k$ over the choice of random oracle. However, [11] uses the random oracle for two purposes: to provide a collision-intractible hash function, and to provide a (publicly accessible) pseudorandom function, which the prover can use to simulate the "sampling verifier". Because we are working in a model with a polynomially-bounded prover (Alice or Bob), we can replace the random oracle hash function with a collision-resistant hash function; the existence of such functions is implied by the hardness of discrete logarithms[7], which is in turn implied by the DDH assumption. The publicly-accessible pseudorandom function can be replaced by an interactive evaluation (using the covert version of Yao's protocol) of a pseudorandom function with the key chosen by the verifier.

**Using CS Proofs to prove that $G_i$ computes $C_i$** Let $M$ denote the TM that computes $C_i$ on input $1^k$, $i$, and $C_f$. (Such machines must exist, since Alice and Bob are uniform. $C_f$ is allowed as input so that nonuniform functionalities can be included) Let $M'$ denote a machine which on input $K \leftarrow U_k, C_f, i$, runs $M(1^k, i, C_f)$ to produce $C_i$ and then garbles $C_i$ using a pseudorandom generator keyed by $K$ to produce $G_i$. Let $M''(K, C_f, i)$ denote the machine that runs $M'(K, C_f, i)$ and computes the collision-resistant hash $h(G_i)$ as its output. Let $T_{i,f} = TIME(M''(\cdot, C_f, i))$, then it is evident that $T_{i,f} \leq k^c \times |C_i|$, for some constant $c$.

The circuit $C_{i+1}$ is a pair of circuits $(F, V)$. $F$ computes a pseudorandom function $F_K : \{0,1\}^k \to \{0,1\}^k$. $V$ computes the verifier for a CS Proof of the statement $(M'', x||C_f||i, h(G_i), T_{i,f})$ using $F_K$ as its random oracle, and also verifies the state $S_i$ and commited bit $b_i$, and if the proof, state, and commited bit verify, outputs $S_{i+1}$ and $b_{i+1}$ as in the previous construction.

It is clear that there is a constant $d$ such that $|C_{i+1}| \leq (|M''| + 2k + |C_f| + \log i + \log T_{i,f})^d$. Choosing $c > 1$ such that $|C_f| \leq k^{c-\epsilon}$, we have $|C_{i+1}| \leq (|M''| + k^c + \log |C_i|)^d$, and we can verify by induction that for sufficiently large $k$, $|C_i| \leq (2k^c)^d$, which is $poly(k)$, as required.

By construction, this protocol still satisfies the required security properties: if $P_\sigma$ aborts with significant advantage in distinguishing whether $P_{1-\sigma}$ has run the protocol and the functionality $f$ is fair, $P_{1-\sigma}$ also has significant advantage; and if $P_\sigma$ gives incorrect inputs at round $i$, he gains no significant advantage over aborting at round $i$. The security of Micali's CS Proof scheme implies that $P_{1-\sigma}$ still cannot cheat by giving an incorrect proof, except with negligible probability.

## References


1. L. von Ahn and N. Hopper. Public-Key Steganography. IACR e-print archive report 2003/233, 2003.
2. M. Backes and C. Cachin. Public-Key Steganography with Active Attacks. IACR e-print archive report 2003/231, 2003.
3. M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. *Advances in Cryptology – Proceedings of CRYPTO '89*, LNCS 435, pp. 547–557, 1990.
4. C. Cachin. An Information-Theoretic Model for Steganography. In: *Information Hiding, 2nd International Workshop*, LNCS 1525, pp 306–318, 1998.
5. D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology* 1(1), pages 65-75, 1988.
6. D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM* 24(2), pages 84-88, 1981.
7. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer. In: *Advances in Cryptology – CRYPTO '91*, pp 470–484, 1991.
8. N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. *Proceedings of the 11th USENIX Security Symposium*, 2002.
9. J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. *A pseudorandom generator from any one-way function*. SIAM Journal on Computing, 28(4):1364–1396, 1999.
10. N. Hopper, J. Langford and L. Von Ahn. Provably Secure Steganography. *Advances in Cryptology – Proceedings of CRYPTO 2002*, LNCS 2442, pp. 77–92.
11. S. Micali. CS Proofs. In: *Proceedings of 35th FOCS*, pp 436–453, IEEE Computer Society Press, 1994.
12. M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings, 1999 ACM Conference on Electronic Commerce.*
13. O. Goldreich. Secure Multi-Party Computation. Manuscript. *http://philby.ucsd.edu/books.html*. 1998.
14. O. Goldreich, S. Goldwasser and S. Micali. How to construct pseudorandom functions. In: *J. of the ACM.*, vol 33. 1998.
15. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. In: Nineteenth Annual ACM Symposium on Theory of Computing. Pages: 218 - 229.
16. L. Reyzin and S. Russell. Simple Stateless Steganography. IACR e-print archive report 2003/093, 2003.
17. M. Waidner. Unconditional sender and recipient untraceability in spite of active attacks. *Advances in Cryptology: EUROCRYPT'89*, pages 302-319, 1989.
18. A. C. Yao. How to Generate and Exchange Secrets. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pages 162–167.