A Minimal Model for Secure Computation*

Uri Feige[†] Joe Kilian[‡] Moni Naor[§]

Abstract

We consider a minimal scenario for secure computation: Parties A and B have private inputs x and y and a shared random string r. A and B are each allowed to send a single message to a third party C, from which C is to learn the value of $f(x, y)$ for some function f, but nothing else. We show that this model is surprisingly powerful: every function f can be securely computed in this fashion. If the messages are required to be of polynomial size, then we exhibit an efficient protocol for any function f computable in nondeterministic logspace. Using a computational notion of security, we exhibit efficient protocols for any polynomial-time computable function f, assuming the existence of one-way functions. The above results generalize to the case where there are more than two parties with private inputs.

The minimalistic nature of our model makes it easy to transform positive results achieved in our model to other more general models of secure computation. It also gives hope for lower-bound proofs. We give an alternative characterization of our model in terms of graph embeddings, and use this to show that for most Boolean functions on $\{0,1\}^n \times \{0,1\}^n$, the need to hide just one of the input bits from C requires a communication overhead of n bits.

1 Introduction

Suppose that two parties, Alice with input x and Bob with input y, wish to jointly compute a function $f(x, y)$ of their inputs without revealing more information about their inputs than necessary (i.e the value of $f(x, y)$). As stated this is an impossible task for any "interesting" function f. For instance, Chor and Kushilevitz [8] showed that the only Boolean functions computable in this way are those of the form $f(x, y) = f_a(x) \oplus f_b(y)$. Furthermore, even if one is willing to rely on cryptographic assumptions, then Kilian [14] has shown that securely computing any function with the "imbedded or" property[1] yields the ability to perform *Oblivious Transfer* (OT). Note that OT is a stronger (in the sense of [12]) assumption than the existence of one-way functions.

In this paper we investigate a "minimal" extension of the two party scenario: we add a trusted party Carol who should do the computation. The communication pattern is minimal: Alice and Bob agree on (or are given) a secret random string; they each send Carol a single message which is a function of their input and the random string. Based on the two messages Carol computes and announces the result. We would like Carol to be completely in the dark as to the inputs of Alice and Bob.

---

*This is a slightly revised version of the STOC 1994 paper.

[†]Dept. of Applied Math. and Computer Science, Weizmann Institute, Rehovot 76100, Israel. E-mail: feige@wisdom.weizmann.ac.il.

[‡]NEC Research Institute, Princeton, NJ 08540, USA. joe@research.nj.nec.com.

[§]Dept. of Applied Math. and Computer Science, Weizmann Institute, Rehovot 76100, Israel. E-mail: *naor@wisdom.weizmann.ac.il*. Research supported by an Alon Fellowship.

[1]A function $f(x, y)$ possesses an imbedded or if there are $x_0, y_0, x_1, y_1, v_0, v_1$ such that for $i, j \in \{0,1\}$i, $f(x_i, y_j) = v_{i \vee j}$

Additional motivation for our model comes from the problem of encrypting audio conference calls [19, 6, 11]. Heiman [11] studied a method where the bridge (i.e. the apparatus connecting the callers) is not to be trusted with the secret information, but nevertheless should determine which participant is "talking the loudest" and transmit his or her encrypted voice to the others. Carol (the bridge) should therefore compute the max function. Such a scenario is captured by our model. The appendix further discusses this specific problem, and offers an elegant protocol for it.

In this paper we outline general principles for computing arbitrary functions in our model. We obtain the following results:

- Any function f can be computed securely. The amount of communication may however be exponential.
- Every function computable in non-deterministic logspace has an *efficient* secure protocol; the amount of communication, number of random bits and the internal computations are all polynomially bounded.
- Assuming one-way functions exist there is an efficient computationally secure protocol for any function in P.

We also show a lower bound on the complexity of securely computing random Boolean functions: for most Boolean functions on $\{0,1\}^n \times \{0,1\}^n$, the parties must send n extra bits, even if they only want to hide one of the input bits from Carol.

In the rest of this section we survey related work, and provide precise definition of our model and security requirements. Sect. 2 shows the construction of secure protocols for general functions. Sect. 3 shows our efficient construction of secure protocols for non-deterministic logspace computable functions. Sect. 4 provides the computationally secure solutions based on one-way functions and Sect. 5 proves our lower bound. The appendix contains some elegant protocols for special functions.

1.1 Related work

In some respects, our model of secure computation resembles that of *instance hiding* [1]. In the instance hiding scenario, a computationally limited party A that holds input x uses a computationally unlimited party C in order to compute $f(x)$, without C learning anything about x (except for its length, and the value of $f(x)$). Thus in both models, party C must not learn the input. However, in each model C is used for a different reason. In our secure computation model, parties A and B each have the power to compute f, but they need C because neither one of them holds the whole input. Our model is most interesting and has potential cryptographic applications if f is an easily computable function. In the instance hiding model, party A holds the input, and cannot compute $f(x)$ because of lack of computational power. Hence [1] analyse their model in cases that f is not known to be computable in polynomial time. Their main result is that NP-hard problems, such as SAT, do not have instance hiding schemes unless the polynomial hierarchy collapses. The same result holds for our secure computation model, if we restrict A and B to random polynomial time computations.

The instance hiding scenario has been extended to the multiple-oracle scenario, with the surprising result that any function has an instance hiding scheme with polynomially many oracles [4]. The main open question regarding instance hiding is whether a constant number of oracles suffice. It is interesting to note that the secure computation model relates to this question. If in our model, all functions (within a given complexity class) can be securely computed with polynomially many shared random bits and polynomial message size (and no restriction on the computational power of A and B), then all functions have a 3-oracle 2-round instance hiding protocol. (The verifier splits its input into the sum of two random inputs. In the first round, one oracle gets to compute M_A and another gets to compute M_B. In the second round, the third oracle computes the function, based on the messages M_A and M_B.)

Our model is also related to the problem of fault tolerant distributed computing [5, 7]. Most notably, our efficient protocol for computing any NLOGSPACE function can be used in order to extend the result of [3], that any function in (algebraic) NC^1 can be computed efficiently in a fault tolerant manner in a constant number of rounds. (Details are omitted from the current abstract.)

There are several natural extensions to our model. In one of them, the input is distributed among k players, who want Carol to compute f for them. Our results extend to the multiparty case. This issue is touched upon in the appendix.

Another extension is to the case that parties are dishonest, and try to deviate from their protocol. In our minimalistic communication scenario, parties have very little opportunity of doing so, and it is relatively simple to make our protocols fault tolerant (see appendix).

1.2 Definitions

Let f be a function $\mathcal{A} \times \mathcal{B} \mapsto \mathcal{D}$ where $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$ are finite domains. A protocol for evaluating f securely consists of $\mathcal{R}$, the collection of random strings, $\mathcal{M}_A$ and $\mathcal{M}_B$, two messages domains and three function f_A, f_B and f_C where $f_A : \mathcal{A} \times \mathcal{R} \mapsto \mathcal{M}_A$, $f_B : \mathcal{B} \times \mathcal{R} \mapsto \mathcal{M}_B$ and $f_C : \mathcal{M}_A \times \mathcal{M}_B \mapsto \mathcal{D}$.

For the protocol to be *perfectly secure* it must obey the following two conditions :

- Correctness - for all $a \in \mathcal{A}, b \in \mathcal{B}$ and $r \in \mathcal{R}$: $f(a,b) = f_C(f_A(a,r), f_B(b,r))$.
- Privacy - for all pairs of input (a_1, b_1) and (a_2, b_2) such that $f(a_1, b_1) = f(a_2, b_2)$ we have that the distribution of messages $(f_A(a_1, r), f_B(b_1, r))$ where r is chosen at random from $\mathcal{R}$ is identical to the distribution of messages $(f_A(a_1, r), f_B(b_2, r))$ where r is chosen at random from $\mathcal{R}$.

Our notion of privacy is information theoretic. A relaxed computational notion of privacy is presented and used in Sect. 4.

2 Securely computing any function

We show how to securely compute any Boolean function $f : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$. This trivially extends to k valued functions, since each of the $\log k$ output bits represents a Boolean function.

Alice holds input a, Bob holds input b, and Carol needs to compute $f(a,b)$. Represent $f(x,y)$ as a bi-partite graph G_f, x's on the left, y's on the right, and edge between x and y iff $f(x,y) = 1$. Using a shared random string of length $2^n + n$, Alice and Bob modify G_f as follows. With each right hand side vertex y, they associate a random bit r_y. If $r_y = 1$, they complement the edges entering y (edges become nonedges, and nonedges become edges). The leftover n random shared bits, denoted by $\pi \in \{0,1\}^n$, are used in order to choose a cyclic permutation on the right hand side vertices.

- Bob sends the location of b in the cyclic permutation (n bits). That is, $M_B = b - \pi \pmod{2^n}$.]item Alice sends the permuted edge pattern going out of a (2^n bits). That is, Alice sends the following list of bits:

$$M_A = f(a,\pi) \oplus r_\pi \ , \ f(a,\pi+1) \oplus r_{\pi+1} \ , \ldots , f(a,\pi-2) \oplus r_{\pi-2} \ , \ f(a,\pi-1) \oplus r_{\pi-1}$$

where addition and subtraction are done mod 2^n.

- In addition, Bob sends the one bit $M_b = r_b$.

In order to compute $f(a,b)$, Carol looks up the M_Bth entry in the list M_A, and complements it iff $M_b = 1$.
Correctness: the effect of Alice sending the values $f(a,y)$ in permuted order is undone by Bob sending $b-\pi$. The effect of Alice *xoring* the values $f(a,b)$ with r_b is undone by Bob sending M_b.
Privacy: for any a,b, there is a one-to-one correspondence between the random bits π and the message M_b, and for each a,b,π, there is a one-to-one correspondence between the random bits r_y and the message M_A. Hence the distribution of the messages M_A and M_B is uniform, and completely independent of the values of a and b. The value of message M_b is then determined uniquely by the values of M_A, M_B, and $f(a,b)$. Hence the distribution of messages of Alice and Bob depends only on $f(a,b)$, as desired.

The complexity of the protocol: Alice sends 2^n bits, Bob sends $n+1$ bits, and they both share 2^n+n random bits. We therefore have:

Theorem 1 For any function f there exists a perfectly secure evaluation protocol.

3 Efficient solutions for subclass of functions

The solution given above for general functions f requires exponential-sized messages in the worst case. We do not know of any protocol that requires only polynomial-sized messages for all functions f, and conjecture that no such protocol exists. Using the information-theoretic notion of privacy, we exhibit a communication-efficient protocol for any f "computable" in nondeterministic logspace. By computable in nondeterministic logspace, we mean that the language

$$\{(i,x,y) | \text{the } i\text{th bit } f(x,y) \text{ is } 1\}$$

is in nondeterministic logspace.

3.1 Securely computing group products

A useful subclass of functions are those that compute products over finite groups. For this class, we have a simple solution based on randomizing group products, which serves as the basis for our more complicated protocols. First, we review the randomizing technique of [13]. Let G be a finite, possibly noncommutative group, let $x = (g_1, \ldots, g_n)$ and $y = (g_{n+1}, \ldots, g_{2n})$, where $g_1, \ldots, g_{2n} \in G$, and let

$$f(x,y) = g_{i_1} g_{i_2} \cdots g_{i_m},$$

for some sequence $i_1, \ldots, i_m$. We transform the sequence $g_{i_1} \ldots g_{i_m}$ into a random sequence $g'_1, \ldots, g'_m$ that has the same product.

Let $r_1, \ldots, r_{m-1} \in G$ be uniformly distributed. Define r_0 and r_m to be the identity element of G, and for $1 \le j \le m$ define

$$g'_j = r_{j-1}^{-1} g_{i_j} r_j.$$

When we take the product of the g'_j's, the r_j terms cancel out, yielding $g'_1 \cdots g'_m = g_{i_1} \cdots g_{i_m}$. By an inductive argument, one can show that for any sequence $g'_1, \ldots, g'_m$ and any sequence $g_{i_1}, \ldots, g_{i_m}$ such that $g'_1 \cdots g'_m = g_{i_1} \cdots g_{i_m}$, there exists exactly one consistent setting for $r_1, \ldots, r_{m-1}$. Thus, $g'_1, \ldots, g'_m$ will be uniformly distributed over all sequences with the same product as $g_{i_1}, \ldots, g_{i_m}$. From a knowledge perspective, $g'_1, \ldots, g'_m$ reveals $f(x, y)$ and nothing else.

Given their common random bit sequence r, Alice and Bob uniformly generate $r_1, \ldots, r_{m-1}$ in some canonical, deterministic fashion. Alice and Bob's messages to Carol are given by

$$\begin{aligned} M_A &= \{(j, g'_j) | i_j \in \{1, \ldots, n\}\} \text{ and} \\ M_B &= \{(j, g'_j) | i_j \in \{n+1, \ldots, 2n\}\}. \end{aligned}$$

Note that M_A depends only on x and r and M_B depends only on y and r. Furthermore, for all j, g'_j is present in either M_A or M_B. On input (M_A, M_B), Carol computes

$$g'_1 \cdots g'_m = g_{i_1} \cdots g_{i_m} = f(x, y)$$

3.2 The General Construction.

Let $f(x, y)$ be "computable" in nondeterministic logspace as defined above. Without loss of generality we assume that $f(x, y)$ is boolean; otherwise we compute each bit independently. We first reduce computing $f(x, y)$ to computing $s-t$ reachability on $G_1 \cup G_2$ where G_1 is a digraph that depends only on x and G_2 is a digraph that depends only on y. We perform a two-step reduction from this problem to computing a product of nonsingular matrices over a sufficiently large finite field. This product will reveal whether t is reachable from s, but will also yield other information that would destroy the privacy of our protocol. We then perform some additional matrix multiplications to eliminate this extra information.

It is well known that any language in nondeterministic logspace can be reduced to $s-t$ reachability in a directed graph. Thus, given a function f and inputs x and y, we can construct (in logspace) a digraph and nodes s and t such that t is reachable from s iff $f(x, y) = 1$. By inspection of the standard reduction, we note that the instance (G, s, t) produced can be made to have the following properties,

1. Nodes s and t are independent of (x, y), and can be relabeled as desired.
2. For each i and j, the presence of directed edge (i, j) in G is either independent of x and y, depends solely on x or depends solely on y.

We define G_1 as having all the edges of G that are either independent of x and y or depend solely on x, and define G_2 as having all the edges of G that depend solely on y. Thus G_1 can be computed by Alice, G_2 can be computed by Bob, (s, t) can be computed by both Alice and Bob, and $f(x, y) = 1$ iff t is reachable from s in the graph $G = G_1 \cup G_2$.

3.2.1 Converting the graph problem to a matrix product

For convenience, we assume that G_1 and G_2 have n nodes and edge (i, i) is included in G_1 and G_2 (each node has a directed edge to itself). Then if A_i is the adjacency matrix of G_i, the matrix $(A_1 A_2)^n$ provides all the reachability information about G: t is reachable from s iff the (s, t) entry of $(A_1 A_2)^n$ is nonzero. Instead of performing these matrix multiplications over the integers, we treat our entries as elements of $GF[p]$ for p large enough to not involve any "overflows."

A_1 and A_2 are not entirely suitable for our purposes, since for instance they might be singular, so we alter them slightly. Assume without loss of generality that $t = n$. First, we add dummy nodes $n+1, \ldots, 2n-1$ and edges

$$\{(1, n+1), (n+1, 1), (2, n+2), (n+2, 2), \ldots, (n-1, 2n-1), (2n-1, n-1)\}$$

to G_1 and G_2. Note that we do not add edges of the form $(n+i, n+i)$. We then delete all edges of the form $(t = n, i)$ for $t \neq i$, but leave in (t, t). Finally, for convenience we canonically relabel the nodes of G_1 and G_2 so that $s = 1$ and $t = N = 2n-1$. Denote the new graphs obtained by G_1' and G_2', their adjacency graphs by A_1' and A_2', set prime $p > N^N$, and define $H = (A_1' A_2')^N$, where all operation are over $GF[p]$. For the rest of our discussion, we will use the following facts about these matrices, and suppress all other details.

1. There is a path from s to t iff $H[1, N]$ is nonzero. This is true since the dummy nodes do not change the reachability structure of the original nodes.
2. A_1' and A_2' are nonsingular. This is true since their rows can be rearranged so as to make them lower triangular, with 1's along the diagonal.
3. Row N of H is all 0 except for $H[N, N]$, which is 1. This can be shown by induction, and comes from the fact that node $t = N$ is only able to reach itself.
4. H is nonsingular, since it is the product of nonsingular matrices. The $N-1 \times N-1$ upper-left submatrix of H is also nonsingular, which follows by the special nature of row N.

3.2.2 Removing information from the matrix product

Knowing H, one can reconstruct $f(x, y)$, but one might also obtain more information. We therefore multiply H by two other randomly generated matrices that will obliterate all information but whether $H[1, N] = 0$. Let Q' be generated as follows:

1. For $1 \leq i \leq N$, choose $Q'[i, i]$ uniformly from $GF[p] - 0$,
2. For $2 \leq i \leq N-1$ choose $Q'[i, N]$ uniformly from $GF[p]$, and
3. Set all other entries of Q' to 0.

Let Q'' be generated as follows:

1. Choose the $N-1 \times N-1$ upper-left submatrix of Q'' uniformly from the set of nonsingular matrices.
2. For $1 \leq i \leq N-1$ set $Q''[i, N] = Q''[N, i] = 0$.
3. Set $Q''[N, N] = 1$.

One can verify that Q' and Q'' are nonsingular. The following lemma shows that $Q'HQ''$ yields exactly the desired information.

Lemma 1 Let Q', Q'' and H be defined as above, and let $H^* = Q'HQ''$. Then H^* is distributed as follows:

1. The upper-left $N-1 \times N-1$ submatrix of H^* is distributed uniformly over all nonsingular matrices.
2. If $H[1,N]=0$ then $H^*[1,N]=0$, and if $H[1,N] \neq 0$ then $H^*[1,N]$ is distributed uniformly and independently over $GF[p]-0$.
3. For $1 \leq i \leq N-1$, $H^*[N,i]=0$.
4. $H^*[N,N]$ is distributed uniformly and independently over $GF[p]-0$.
5. For $2 \leq i \leq N-1$, $H^*[i,N]$ is distributed uniformly and independently over $GF[p]$.

Lemma 1 implies that one can easily reconstruct $f(x,y)$ from H^* but obtain no other information about (x,y).

Proof: (Sketch) Let $H' = HQ''$. The $N-1 \times N-1$ upper-left submatrix of H' is equal to the $N-1 \times N-1$ upper-left submatrix of H, which is nonsingular, multiplied on the right by a uniformly distributed nonsingular matrix, and hence is also a uniformly distributed nonsingular matrix. The Nth column of H' is equal to the Nth column of H. Finally, for $1 \leq i \leq N-1$, $H'[N,i]=0$.

Now consider $H^* = Q'H'$. The first row of H^* is equal to the first row of H' multiplied by $Q'[1,1]$, a random element of $GF[p]-0$. The last row of H^* is equal to the last row of H' multiplied by $Q'[N,N]$, a random element of $GF[p]-0$. The remaining rows $2 \leq i \leq N-1$ of H^* are equal to the corresponding row of H' multiplied by $Q'[i,i]$ (a random element of $GF[p]-0$) plus the last row of H' multiplied by $Q'[i,N]$, a random element of $GF[p]$.

Therefore we can conclude that: (i) The $N-1 \times N-1$ upper-left submatrix of H^* is a random nonsingular matrix over $GF[p]$ (ii) $H^*[1,N]$ is 0 if $H'[1,N]=H[1,N]=0$ and a random element from $GF[p]-0$, independent of the rest of the matrix, otherwise. (iii) By a straightforward calculation, $H^*[N,i]=0$ for $1 \leq i \leq N-1$. (iv) $H^*[N,N]$ is equal to $H'[N,N](=1)$ times $Q'[N,N]$, a uniformly distributed element of $GF[p]-0$, and will thus be a uniformly distributed element of $GF[p]-0$. (v) For $2 \leq i \leq N-1$, $H^*[i,N] = H'[i,N] + H[N,N] \cdot Q'[i,N] = H'[i,N] + Q'[i,N]$. this is the only entry influenced by $Q'[i,N]$ and will therefore a uniformly distributed element of $GF[p]$ independent of the rest of the matrix. ∎

3.2.3 The final protocol

Given the above discussion, the protocol for Alice, Bob and Carol is straightforward. Using their common randomness r, Alice and Bob generate Q' and Q'' as discussed above. They then uniformly sample matrices $R_0, \ldots, R_{2N+1}$ from the set of $N \times N$ nonsingular matrices over $GF[p]$. On input x, Alice computes A_1' and constructs her message

$$M_A = \left((Q'R_0^{-1}), (R_0 A_1' R_1^{-1}), \ldots, (R_{2N-1} A_1' R_{2N}^{-1})\right).$$

On input y, Bob computes A_2' and constructs his message

$$M_B = \left((R_1 A_2' R_2^{-1}), \ldots, (R_{2N} A_2' R_{2N+1}^{-1}), (R_{2N+1} Q'')\right).$$

Finally, Carol computes $H^* = Q'(A_1 A_2)^N Q''$ and outputs 1 iff $H^*[1,N] \neq 0$. By our previous argument, Carol learns nothing aside from the actual product H^*, which by Lemma 1 reveals nothing but $f(x,y)$. Therefore we have

Theorem 2 For every function $f(x,y)$ computable in nondeterministic logspace there is an efficient perfectly secure protocol. In general, if a function f is computable in $NSPACE(s)$ and s is $\Omega(\log n)$, then there is a perfectly secure protocol for evaluating f where the length of the messages is $2^{O(s)}$).

4 A computationally secure protocol.

Under a computational notion of privacy, we exhibit a communication-efficient protocol for any f computable in polynomial time. Due to space limitations, we give only a high level discussion of our protocol, and omit the proof of correctness. We restrict our discussion to the case where the parties are honest. We show a general method for obtaining resiliency in the appendix.

Definition 1 Let D be a nonuniform polynomial time algorithm (i.e., a circuit family). We say that D is a *distinguisher* for an infinite family of input pairs $((x_1,y_1),(x_2,y_2))$ if for some c,

$$|\Pr_r(D(f_A(x_1,r),f_B(y_1,r))=1)-\Pr_r(A(f_A(x_2,r),f_B(y_2,r))=1)|>\frac{1}{n^c},$$

where n is the size of x_1,y_1,x_2,y_2.

Definition 2 A protocol is *computationally private* if there is no D and infinite family of input pairs $((x_1,y_1),(x_2,y_2))$ with $f(x_1,y_1)=f(x_2,y_2)$, such that D is a distinguisher for the family.

Remark: Another approach to defining computationally private protocols includes a unary security parameter k, and allows one to avoid considering infinite families of inputs. One can embed such a notion in the one we present by a padding argument: given a function $f(x,y)$, consider the function $f'(x01^k,y01^k)=f(x,y)$. Privately computing f' on the infinite family of inputs $\{(x01^k,y01^k)\}$ is analogous to computing $f(x,y)$ with an infinite family of security parameters $\{k\}$.

We now outline a computationally private protocol based on one-way functions in which A and B run in time polynomial in n. The protocol is based on the "garbled circuit" construction introduced by Yao [21]. This construction is based on the existence of cryptographically secure pseudo-random generators, which in turn can be based on the existence of one-way functions [10]. We will use this construction as a black-box, and summarize it as follows.

Let $\{C_n(X,Y)\}$ be a circuit family of size m_n that computes function $F_n:\{0,1\}^n\times\{0,1\}^n\rightarrow\{0,1\}$, and let f be a one-way function. Assuming the existence of a one-way function (against nonuniform adversaries), there exist random polynomial time algorithms called SCRAMBLE(C_n,r), SIMULATE(C_n,r) and EVALUATE$(\hat{C}_n,\hat{X},\hat{Y})$ with the following properties:

1. Given random string r, SCRAMBLE(C_n,r) randomly computes a *garbled circuit* $\hat{C}_n$ and garbled input pairs $((Q_0,Q_1),(R_0,R_1))$, where

$$\begin{aligned} Q_b &= Q_{1,b},\ldots,Q_{n,b} \text{ and} \\ R_b &= R_{1,b},\ldots,R_{n,b} \end{aligned}$$

for $b\in\{0,1\}$. For $X=(x_1,\ldots,x_n)\in\{0,1\}^n$ and $Y=(y_1,\ldots,y_n)\in\{0,1\}^n$ we define

$$\begin{aligned} Q_X &= Q_{1,x_1},\ldots,Q_{n,x_n} \text{ and} \\ R_Y &= R_{1,y_1},\ldots,R_{n,y_n}. \end{aligned}$$

2. EVALUATE$(\hat{C}_n, Q_X, R_Y) = C_n(X,Y)$.

3. Given random string r, SIMULATE(C_n, b, r) randomly computes a *simulated garbled circuit* C_n^* and simulated garbled inputs Q_b^*, R_b^*. If $C_n(X,Y) = b$ then the distributions $(\hat{C}_n, Q_X, R_Y)$ and (C_n^*, Q_b^*, R_b^*) are computationally indistinguishable to nonuniform adversaries. Here we implicitly assume that we have an infinite family of circuits and inputs (C_n, X, Y).

A detailed, rigorous discussion of Yao's garbled circuit technique can be found in [18].

Given the above abstraction, we define $(M_A, M_B)[C_n, X, Y]$ as follows. Given circuit C_n and shared random string r, A and B compute

$$\text{SCRAMBLE}(C_n, r) = \hat{C}_n, ((Q_0, Q_1), (R_0, R_1)).$$

Then M_A and M_B are defined by

$$\begin{aligned} M_A &= \hat{C}_n, Q_{1,x_1}, \ldots, Q_{n,x_n} \text{ and} \\ M_B &= R_{1,y_1}, \ldots, R_{n,y_n}. \end{aligned}$$

On input M_A, M_B, C writes $M_A = \hat{C}_n, Q_X$ and $M_B = R_Y$ and computes

$$C_n(X,Y) = \text{EVALUATE}(\hat{C}_n, Q_X, R_Y).$$

We omit the analysis of this protocol. Roughly, correctness of the protocol follows from the definition of EVALUATE. The privacy of the protocol follows from the existence of the SIMULATE protocol. (M_A, M_B) is equivalent to $(\hat{C}_n, Q_X, R_Y)$. If $C_n(X_1, Y_1) = C_n(X_2, Y_2) = b$ then

$$(\hat{C}_n, Q_{X_1}, R_{Y_1}) \text{ and } (\hat{C}_n, Q_{X_2}, R_{Y_2})$$

will both be indistinguishable from the output of SIMULATE(C_n, b, r), where r is uniformly distributed. Hence, they will be indistinguishable from each other.

5 The lower bound

We lower bound the *communication cost*, $c = \max_{x,y,r}[|f_A(x,r)| + |f_B(y,r)|]$, for perfectly secure computation. Note that without privacy requirements, $\forall f,\ c \leq 2n$, and with privacy, $\forall f,\ c \leq 2^n + n + 1$ (see Sect. 2). We prove a modest lower bound of $c \geq 3n - 4$, which holds for almost any function f. This lower bound only requires the following minimal privacy assumption, that the last bit of x remains private.

Minimal privacy: If $f(x,y) = f(\bar{x},y)$, where $\bar{x}$ denotes x with its last bit flipped, then for any messages z and w,

$$\begin{aligned} & Pr_r[(f_A(x,r) = z) \text{ and } (f_B(y,r) = w)] \\ = \ & Pr_r[(f_A(\bar{x},r) = z) \text{ and } (f_B(y,r) = w)] \end{aligned}$$

5.1 Fair embeddings

Represent the function f by a bipartite graph G_f. The 2^n left-hand side vertices of G_f each represents a single possible input x, and the 2^n right-hand side vertices of G_f each represents a single possible input y. Vertices x and y are connected by a white edge if $f(x,y) = 1$, and by a black edge if $f(x,y) = 1$. Represent player C by a bipartite graph G_C. The left-hand side vertices of G_C each represents a single possible message z, and the right-hand side vertices of G_C each represents a single possible message w. If $f_C(z,w)=1$ then (z,w) is a black edge in G_C, and if $f_C(z,w)=0$ then (z,w) is a white edge E_{G_C} (and if $f_C(z,w)$ is undefined then (z,w) is not an edge at all).

To simplify notation, we treat each shared random string r as a function in the following sense: $r(x)$ denotes $f_A(x,r)$, and $r(y)$ denotes $f_B(y,r)$. From the *completeness* property it follows that each r induces an *embedding* of G_f in G_C. That is, (x,y) is a white (black) edge in G_f iff $(r(x),r(y))$ is a white (black, respectively) edge in G_C. Observe that for almost all functions f, any corresponding embedding r must be one-to-one, as it is unlikely that there exist x_1, x_2 such that for all y, $f(x_1,y)=f(x_2,y)$.

All the random strings r collectively induce a *family* of embeddings. From the *privacy* property of the protocol, it follows that the family is *fair*. That is, for any white edge $e \in G_C$, and for any white edges $e_1, e_2 \in G_f$, $|\{r|r(e_1)=e\}| = |\{r|r(e_1)=e\}|$, and similarly for black edges. We shall use the term *fair embedding* as abbreviation for the term "fair family of embeddings".

Thus we transformed our original question on secure computation to the following problem in extremal graph theory: what is the minimal size of G_C so that bipartite graph G_f has a fair embedding into G_C?

5.2 Universal graphs

We start by analyzing the case that C does not know f. That is, G_C is a universal graph that depends on n alone but not on G_f. In Sect. 2 we showed that $c \leq 2^n+n+1$ suffices in order to compute any function, and that x, y, and f, are kept private by this computation. It is not hard to show that if C does not know f in advance, the above bound on communication is nearly tight, even without any privacy requirements.

Claim 1 Player A receives x and f, player B receives y and f, and they want C to compute $f(x,y)$ (even without hiding f, x, and y). Then $c \geq 2^n$, assuming fixed length encoding of messages.

Proof: Since no privacy is required, the optimal protocol for the players is deterministic. Let $N=2^n$. For any f, the graph G_f is embedded in G_C. There are 2^{N^2} possible graphs G_f. Let U be the number of vertices on the left side of G_C, and let V be the number of vertices on its right side. To address vertices of G_C, A needs to send $\log U$ bits, and B needs to send $\log V$ bits (we assume fixed length encoding of messages). There are $U^N V^N$ possible ways of mapping the vertices of G_f to the vertices of G_C. It follows that $(UV)^N \geq 2^{N^2}$, or $\log U + \log V \geq N = 2^n$.

5.3 Hiding a single bit

We return to the question of embeddings when G_C depends on G_f, for G_f a random bipartite graph. We prove our lower bound on the number of communication bits under the minimal privacy assumption: that C does not learn the least significant bit of x. C is allowed to learn all other bits of x, and all bits of y (and A, B, and C, know f in advance).

Definition 3 For vertex x, let $\bar{x}$ denote the *complement* vertex obtained by flipping the least significant bit of x. A colored edge (x,y) is *dangerous* if it is of the same color as its complement edge $(\bar{x},y)$.

Observe that for a random function f, about half the edges of G_f are dangerous.

Definition 4 G_f has a family $R = \{r_i\}$ of embeddings into G_C. Consider two embeddings r_i and r_j, and a dangerous edge $(x, y) \in G_f$. A *trivial match* of r_i and r_j on (x, y) occurs if $r_i(x, y) = r_j(x, y)$. A *nontrivial match* occurs if $r_i(x, y) = r_j(\bar{x}, y)$.

Lemma 2 For most bipartite graphs G_f on $2N$ vertices, for any two embeddings r_i and r_j, the number of nontrivial matches is at most $2N$.

Proof: As noted earlier, we can assume that both r_i and r_j are one-to-one. The nontrivial matches are formed by r_i and r_j agreeing on where in G_C to map a subset $Y \in G_f$ of y vertices, and by agreeing where in G_C to map a subset $X \in G_f$ of x vertices (where r_i maps X and r_j maps the complement of X). There are 2^N possible ways of choosing X, and 2^N ways of choosing Y. Observe that to have $2N$ nontrivial matches, $|X| \cdot |Y| \geq 2N$. Now we use a probabilistic argument. Decide on a random truth table for f. Each nontrivial match has probability $1/2$, independently, of being dangerous. If it is not dangerous, then either r_i or r_j do not form a legal embedding. Hence the probability that any particular choice of X and Y is legal is at most 2^{-2N}.

Lemma 3 For any $k \leq |R|$, there exists $R' \subset R$ of size $|R'| = k$, such that the total number of trivial matches induced by members of R' is smaller than the total number of nontrivial matches induced by members of R'.

Proof: Consider an arbitrary colored edge $e \in G_C$, and an arbitrary dangerous edge $(x, y) \in C_f$. By the minimal privacy property, if there are t members of R that map (x, y) onto e, then there are t members of R that map $(\bar{x}, y)$ onto e. Now select R' at random, and for any x, y, e compute the expectation $\mathcal{E}[\delta]$, where δ is the number of nontrivial matches induced by (x, y) and $(\bar{x}, y)$ on e, minus the number of trivial matches induced by either (x, y) or $(\bar{x}, y)$ on e. It is not hard to see that as each new member of R' is chosen, the expectation $\mathcal{E}[\delta]$ increases. Then, by linearity of expectation

$$E_{R'}[\sum_{x,y,e} \delta] \geq 0$$

and hence for some choice of R' the lemma holds.

Lemma 4 The number of colored edges in G_C is at least $N^3/16$.

Proof: We bound $|E_C|$ by using the fact that embeddings have small nontrivial intersections (Lemma 2), and even smaller trivial intersections (Lemma 3).

Clearly, $|R| \geq 1$. Pick at random r_1. Since every edge of G_C that is an image of a dangerous edge of G_f has to contain a nontrivial match, and since any r_i contributes at most $2N$ nontrivial matches, it follows that $|R| \geq N/4$ (by the assumption that G_f has $N^2/2$ dangerous edges).

Consider R' embeddings chosen from R, where $|R'| = N/8$. By lemmas 2 and 3, and using the first two terms of the inclusion-exclusion formula, we lower-bound E'_C, the number of edges in G_C that are images of dangerous edges of G_f under an embedding $r_j \in R'$. $|E'_C| \geq N^2|R'|/2 - 2N|R'|^2 \geq N^3/16$.

Remark: If x and y are required to be hidden from C, then the argument can be extended to show that the number of random bits shared by A and B is at least $(3n - 5)$, since each colored edge of G_C must be covered $N^2/2$ times.

References

[1] M. Abadi, J. Feigenbaum, J. Kilian, *On Hiding information From an Oracle*, JCSS 39 (1989), 21–50.

[2] D. A. Barrington, *Bounded-width polynomial-sized branching programs recognize exactly those languages in NC^1*, J. of Computer and Systems Sciences, Vol 38, 1988, pp. 150–164.

[3] J. Bar Ilan, D. Beaver, *Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds of Interaction.*

[4] D. Beaver, J. Feigenbaum, *Hiding Instances in Multi-Oracle Queries*, Proc. of 7^{th} STACS, pp. 37–48, Lecture Notes in Computer Science, Vol. 415, 1990.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*, Proc. 20th Symp. on Theory of Computing, 1988, pp. 1–10.

[6] E. Brickell, P.J. Lee and Y. Yacobi, *Secure Audio Teleconference*, Advances in Cryptology - Crypto 87 Proceedings, Springer Verlag, 1987, pp. 418–426.

[7] D. Chaum, C. Crépeau, and I. Damgård, *Multiparty Unconditionally Secure Protocols*, Proc. 20th Symp. on Theory of Computing, 1988, pp. 11–19.

[8] B. Chor and E. Kushilevitz, *A zero-one law for Boolean privacy*, Proc. 21st ACM Symp. on Theory of Computing, 1989, pp. 61–72.

[9] O. Goldreich, M. Micali, A. Wigderson, *How to play any mental game*, Proc. 19th ACM Symp. on Theory of Computing, 1987, pp. 218–229.

[10] J. Hastad, R. Impagliazzo, L. Levin, M. Luby. *Pseudo-random Generation from Oneway Functions,* SIAM J. Computing 28(4), 1999, pp. 1364–1396.

[11] R. Heiman, *Secure Audio Teleconferencing: A Practical Solution*, Advances in Cryptology - EUROCRYPT '92 Proceedings, Lecture Notes in Computer Science 658, Springer, 1993, pp. 437–448.

[12] R. Impagliazzo and S. Rudich, *Limits on the provable consequence of one-way permutations*, Proc. 21st ACM Symp. on Theory of Computing, 1989, pp. 44–61.

[13] J. Kilian, **Use of Randomness in Algorithms and Protocols**, MIT Press, Cambridge, Massachusetts, 1990.

[14] J. Kilian, *A general completeness theorem for two-party games*, Proc. 23rd ACM Symp. on Theory of Computing, 1991, pp. 553–560.

[15] E. Kushilevitz, *Privacy and communication complexity*, Proc. of the 30th IEEE Symp. on Foundations of Computer Science, 1989, pp. 416–421.

[16] A. Shamir, R. Rivest and L. Adleman, *Mental Poker*, in **The Mathematical Gardner**, edited by David Klarner, Wadsworth International, Belmont, 1981, pp. 37–43.

[17] T. Rabin and M. Ben-Or. *Verifiable Secret Sharing and Multiparty Protocols with Honest Majority*, Proc. 21st ACM Symp. on Theory of Computing, 1989,

[18] P. Rogaway. *Phd. Thesis.* MIT.

[19] D.G. Steer, L. Strawscynski, W. Diffie and M. Wiener, *A Secure Audio Teleconference System*, Advances in Cryptology - Crypto 88 Proceedings, Springer Verlag, 1988, pp. 520–528.

[20] A. C. Yao, *Protocols for Secure Computations*, Proc. of the 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp. 160–164.

[21] A. C. Yao, *How to Generate and Exchange Secrets*, Proc. of the 27th IEEE Symp. on Foundations of Computer Science, 1986, pp. 162–167.

A Securely computing the maximum of two numbers

The problem of securely computing the maximum of two numbers arises in the context of secure audio teleconferencing. Heiman [11] studied a method where the bridge (i.e. the apparatus connecting the callers) is not to be trusted with the secret information, but nevertheless should determine which participant is "talking the loudest" and transmit his or her encrypted voice to the others. Carol (the bridge) should therefore compute the max function. It is clear that in this problem it is very important to keep the communication pattern as simple as possible and also to minimize the amount of bits each participant must send. To this end Heiman adopts a solution of [6], which is efficient, but does reveal partial information (the difference between the two voice signals) to Carol. Heiman asks if there is a more secure method of comparing encrypted numbers. We present an elegant (though not necessarily practical) solution.

The inputs to players A and B are $x_A, x_B \in \{0,1,2\}$ and they want to determine which of the two numbers is larger. Consider the comparison function:

$$COMP(x_A, x_B) = \begin{cases} 1 & \text{if } x_A > x_B \\ -1 & \text{if } x_A < x_B. \\ 0 & \text{if } x_A = x_B. \end{cases}$$

The random string that A and B agree on consists of (r_1, r_2) where $0 \leq r_1 < 7$ and $r_2 \in \{1,2,4\}$. Player A computes and sends $M_A = r_1 + r_2 \cdot x_A \pmod 7$ and player B commutes and sends $M_B = r_1 + r_2 \cdot x_B \pmod 7$. Carol determines the result according to the following:

$$COMP(x_A, x_B) = \\ \begin{cases} 0 & \text{if } M_A - M_B = 0 \\ 1 & \text{if } M_A - M_B \bmod 7 \in \{1,2,4\} \\ -1 & \text{if } M_A - M_B \bmod 7 \in \{3,5,6\} \end{cases}$$

Correctness follows by noting that $M_A - M_B = r_2(x_A - x_B) \bmod 7$. The quadratic residues $\bmod 7$ are $\{1,2,4\}$. Hence $r_2(x_A - x_B) \bmod 7$ is a quadratic residue iff $x_A - x_B$ is a quadratic residue $\bmod 7$. Incidently[2] this happens precisely when $x_A > x_B$. Security follows by noting that for every $x_A \neq x_B$ and every pair of messages M_A, M_B yielding the right result there is a unique r_1, r_2 that maps x_A and x_B to M_A and M_B.

[2] This is not true in general of course, however if we want to compare two elements from a domain of size k, then by Weil's Theorem for a sufficiently large prime p we will have a stretch of k quadratic non-residues followed by k quadratic residues mod p and thus we can run an algorithm similar to the one described here.

B Securely computing the logical and of k bits

This problem arises when we extend our model to k parties who want to securely compute a k input function. The players agree of a prime $p > k$, on r, $0 < r < p$, and on $r_1, \dots, r_k$, such that $\sum_i r_i = 0 \pmod{p}$. Each player sends $m_i = r(1 - x_i) + r_i \pmod{p}$. C decides 0 iff $\sum m_i = 0 \pmod{p}$.

(Proof of correctness and privacy omitted.)

This can be used in order to securely compute any k-input function, by reduction to the two party protocol of Sect. 2. One player plays the role of A, and the other $k-1$ players jointly comprise player B, who needs to compute a pointer M_B. This is done by not representing the pointer as a binary string, but as an indicator function (one 1-entry among a long list of 0-entries, indication the location to which the pointer points). Computing each entry reduces to computing logical and of $k-1$ bits, where each bit specifies if the input of the corresponding player allows this entry to be the correct one. The bit M_b can be computed as by taking bitwise and of the indicator vector with a vector of the y_b.

C Achieving resilience against malicious adversaries:

We address here the two party case. In this case, nothing can be done if two of the players form a coalition in order to cheat. Hence we remain with two possible forms of cheating.

As our model is stated, C cannot actively violate the protocol since she is not supposed to send any messages. However, in a natural extension of the protocol, we may require C to announce the result. We would like to make it impossible for C to announce a wrong result. This can achieved by having A and B secretly agree on some random authentication code for the output, and requesting C to compute the function f composed with the function representing the authentication code. Observe that in all our constructions, C does not learn which function she is computing, and hence the authentication code remains secret. The is negligible probability that C would guess a valid codeword (other than the true output) to announce as the result.

Neither A nor B can unfairly receive any information, since they do not receive any messages. However, either A or B can control the value of $f(x, y)$ computed by C by giving a value of M_A or M_B that does not correspond to a legal input. Note that C cannot determine this by herself, since she does not know A and B's shared random string.

First, we note that by a simple rearrangement of the bits in M_A and M_B, all of our protocols can be put into the following form:

1. Based on random string r, both A and B can compute strings

$$(\mathcal{A}_1^0, \mathcal{A}_1^1), \dots, (\mathcal{A}_n^0, \mathcal{A}_n^1) \text{ and } \\ (\mathcal{B}_1^0, \mathcal{B}_1^1), \dots, (\mathcal{B}_n^0, \mathcal{B}_n^1).$$

2. If $X = x_1, \dots, x_n$ and $Y = y_1, \dots, y_n$, then

$$\begin{aligned} M_A &= \mathcal{A}_1^{x_1}, \dots, \mathcal{A}_n^{x_n} \text{ and} \\ M_B &= \mathcal{B}_1^{y_1}, \dots, \mathcal{B}_n^{y_n}. \end{aligned}$$

Thus, for example A knows that B should send either $\mathcal{B}_i^0$ or $\mathcal{B}_i^1$, and wants C to abort if he receives some other string. However, A doesn't want C to obtain any information about the value of the string that B did not send. Given a security parameter k, we show how A (B) can ensure that B (A) will be discovered with probability $1 - 2^{-k}$ if he sends an incorrect message. The value of k must be decided on ahead of time, and causes a $poly(k)$ overhead in the message sizes of our protocols.

We use a "fingerprinting" technique reminiscent of [17] and very similar to that in [13]. To represent a bit b, randomly construct a sequence of pairs $\vec{b} = ((b_1^0, b_1^1), \ldots, (b_k^0, b_k^1))$ such that $b = b_i^0 \oplus b_i^1$ for $1 \leq i \leq k$. Given a bit sequence $c = (c_1, \ldots, c_k) \in \{0,1\}^k$, define the fingerprint $f_c(\vec{b})$ as the sequence $b_1^{c_1}, \ldots, b_k^{c_k}$. By a straightforward probability argument, if $\vec{b}$ and c are chosen uniformly, and one does not know c, if one generates a legal representation $\vec{b}'$ for $1 - b$, then $f_c(\vec{b}) \neq f_c(\vec{b}')$ with probability $1 - 2^{-k}$. By legal, we mean the each pair in $\vec{b}'$ sums to the same value mod 2. We represent a string $B = b_1, \ldots, b_m$ by choosing $\vec{b}_i$ as before and setting $\vec{B} = \vec{b}_1, \ldots, \vec{b}_k$. We define $f_c(\vec{B}) = f_c(\vec{b}_1), \ldots, f_c(\vec{b}_m)$.

Using their shared randomness, A and B compute

$$(\vec{\mathcal{A}}_1^0, \vec{\mathcal{A}}_1^1), \ldots, (\vec{\mathcal{A}}_n^0, \vec{\mathcal{A}}_n^1) \text{ and } (\vec{\mathcal{B}}_1^0, \vec{\mathcal{B}}_1^1), \ldots, (\vec{\mathcal{B}}_n^0, \vec{\mathcal{B}}_n^1).$$

A chooses $c_a \in \{0,1\}^k$ and $s_{a,1}, \ldots, s_{a,m} \in \{0,1\}$ using private coins (not known by B). Similarly, B uses private coins to choose $c_b \in \{0,1\}^k$ and $s_{b,1}, \ldots, s_{b,m} \in \{0,1\}$. We now define our messages by

$$\begin{aligned} M_A = \vec{\mathcal{A}}_1^{x_1}, \ldots, \vec{\mathcal{A}}_n^{x_n}, (f_{c_a}(\vec{\mathcal{B}}_1^{s_{a,1}}), f_{c_a}(\vec{\mathcal{B}}_1^{1-s_{a,1}})), \ldots, \\ (f_{c_a}(\vec{\mathcal{B}}_n^{s_{a,1}}), f_{c_a}(\vec{\mathcal{B}}_n^{1-s_{a,n}})), c_a \end{aligned}$$

$$\begin{aligned} M_B = \vec{\mathcal{B}}_1^{y_1}, \ldots, \vec{\mathcal{B}}_n^{y_n}, (f_{c_b}(\vec{\mathcal{A}}_1^{s_{b,1}}), f_{c_b}(\vec{\mathcal{A}}_1^{1-s_{b,1}})), \ldots, \\ (f_{c_b}(\vec{\mathcal{A}}_n^{s_{b,1}}), f_{c_b}(\vec{\mathcal{A}}_n^{1-s_{b,n}})), c_b. \end{aligned}$$

C tests each message purporting to be $\vec{\mathcal{A}}_i^{x_i}$ by first checking that it is a legal encoding for a string and then computing $f_{c_b}(\vec{\mathcal{A}}_i^{x_i})$ and checking to see that it is equal to either $f_{c_b}(\vec{\mathcal{A}}_i^{s_{b,i}})$ or $f_{c_b}(\vec{\mathcal{A}}_i^{1-s_{b,i}})$. If it passes this check, then C computes $\mathcal{A}_i^{x_i}$. C processes $\vec{\mathcal{B}}_i^{y_i}$ analogously.

With the above modification, neither party can send C an invalid string without being caught with probability $1 - 2^{-k}$. However, A (symmetrically B) can give incorrect fingerprints that will cause C to improperly abort the protocol with a probability that depends on B's input. This may indirectly allow A to learn information about B's input (if C publicizes his decision to abort) and can be used to unfairly influence the output of the protocol. For example if C is to compute $x \oplus y$, then A can make C abort iff $x \oplus y = 1$. However, there is an easy trick for dealing with this problem (cf. [13]). Given a function $f(x_1, \ldots, x_n)$ (we don't distinguish between each party's input bit), consider the function

$$f_k((x_{1,1}, \ldots, x_{1,k}), \ldots, (x_{n,1}, \ldots, x_{n,k}))$$

that computes $x_i = x_{i,1} \oplus x_{i,k}$ and then computes $f(x_1, \ldots, x_n)$. Each party randomly and privately expands each of their input bits into an exclusive-or of k bits, and then securely evaluates f_k. One can show that no matter what strategy a malicious player uses, the probability that C will abort will vary by an amount $O(n2^{-k})$ regardless of each party's original input.